AN ABSTRACT OF THE THESIS OF

William S. Donaldson for the degree of Master of Science in Crop and Soil Science presented on November 14, 1991.

Title: Integrating Real-Time Weather Data with Dynamic Crop Development Models

Abstract approved: Redacted for Privacy

Dale N. Moss

Crop development models are commonly used in research. However, their use as crop management tools for growers is rare. Decision support systems (DSS), which combine crop models with expert systems, are being developed to provide management assistance to growers. Researchers at Oregon State University are in the process of developing a DSS. Research was conducted to develop a computer program to provide current and generated weather data for use by the DSS. The objectives of this research were to obtain a weather station, develop a set of quality control procedures to check data from the station, obtain a weather generator program, and create a weather data manager program to implement the above objectives.

A weather station was obtained and was placed near two existing weather stations for ten months. Data from the weather station was compared with the other two stations for values of monthly average maximum temperature, minimum

temperature, and daily total solar radiation and monthly total precipitation. The weather station performed well. Only measurements of total daily solar radiation were consistently different from the other stations. Based on a comparison of the weather station with an Eppley pyranometer, a factor was calculated to correct the solar radiation readings.

The quality control procedures used on the weather data were adapted from automated procedures given in the literature. When tested, the procedures performed as desired. When used on actual data from the weather station, values that failed the procedures were apparently legitimate values. Options were added to the data manager program that allow the user to quickly decide what to do with failed values.

For a weather data generator, WGEN was chosen from the generators presented in the literature. An input parameter file was created for the Corvallis, Oregon area and thirty years of data were generated. Monthly means from this data were compared with thirty-year historical monthly means for Corvallis. Precipitation data from WGEN compared well with the historical data. The generated data for maximum and minimum temperature and daily total solar radiation had great differences from the historical data. It is believed that the input parameters for the Corvallis area suggested by the authors of WGEN are not appropriate.

The weather data manager program was written in the C

programming language, and occupies approximately 98 kilobytes of disk space, not including the eleven files created directly and indirectly by the program. The main functions of the program are: 1) retrieving data from the weather station and performing quality control procedures on the data (allowing the user to decide what to do with values that failed QC); 2) viewing and editing of files by the user; 3) weather data generation (creating a file of only generated data or appending generated data to the file of current data from the weather station to create a file containing a full year of weather data); and 4) miscellaneous functions (monitoring the weather station, setting the calendar in the station's datalogger, and changing information used by the data manager program).

It is hoped that this program will be a significant contribution towards the development of a decision support system.

Integrating Real-Time Weather Data with Dynamic Crop Development Models

by

William S. Donaldson

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed November 14, 1991

Commencement June 1992

APPROVED:

Redacted for Privacy

Professor of Crop and Soil Science in charge of major

Redacted for Privacy

Head of department of Crop and Soil Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented November 14, 1991

Typed by and for William S. Donaldson

ACKNOWLEDGEMENTS

I would like to thank Dr. Dale Moss for his assistance and for sharing with me his incredible knowledge of crop physiology. I thank Dr. John Bolte for his help with my many questions about programming. I also thank Dr. Glenn Klein and Dr. Larry Lev for their assistance and for being on my committee.

Many thanks to Kevin Currans and George Taylor for providing the necessary weather data for the quality control procedures and the station comparisons.

I would like to thank the Department of Crop and Soil Science for providing the funding for this research.

My wife, Amy, deserves special thanks for being patient and supportive as this thesis kept getting larger and larger and kept taking longer and longer. I must also thank my Lord Jesus Christ for providing His grace, strength, and discipline throughout this time.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTEGRATING REAL-TIME WEATHER DATA WITH DYNAMIC CROP DEVELOPMENT MODELS

INTRODUCTION

Crop models are sets of mathematical formulas used to describe the response of a crop to its environment. The development and use of crop models have increased rapidly in recent years (Day, 1984). Whisler, et al. (1986) give three reasons for developing crop models:

1) to aide interpretation of experimental results. Such models are usually single equations that help scientists understand and quantify various agronomic processes.

2) to allow a researcher to predict results if a change is made in some part of the agronomic system. Models used for this purpose can help a researcher decide which field tests might be worth the expense of conducting.

3) to give growers a research-based tool for crop management. Such a tool can assist a grower in making use of the environment and of the inputs to a crop more efficiently. "What if...?" simulations could be performed with the model, allowing a grower to determine the profitability of different management options.

While crop models are widely used in research, their use as crop management tools is rare. In fact, models are

often considered as simply research tools, not having practical applications (Legg, 1981). Jones, et al. (1987) provide an explanation of why models are not used more as "real-world" management tools: 1) models "fall short" of the goal of predicting growth and yield for "any soil, climate and management regime"; 2) models are certain to omit "various environmental, soil and biological characteristics" that influence crop growth and yield; and 3) only people who are "very familiar with the specific capabilities and limitations" of the models can effectively interpret the models' results.

Crop models have been improved and combined with other technology to overcome the above limitations. In regard to the first limitation listed above, some models presently exist that are usable over a wide range of environment and management regimes. Examples are the CERES models for wheat and maize. These models were developed and validated using many and varied experimental data sets, resulting in widely adaptable models.

The second limitation models have of greatly simplifying crop growth is also being overcome. Several models exist today, such as the CERES models, that take into account most, if not all major factors presently known to influence crop growth and yield, except for pest and disease factors. These models simulate the growth of a crop by theoretically developing roots, leaves, grain, and other

plant organs. These models are able to predict dynamic changes in different parts of the crop system (such as leaf area or soil water content) over the course of the growing season (Whisler, et al., 1986). Rapidly advancing computer technology has enabled these models to become more complex and still be able to simulate an entire growing season in a matter of seconds. These models are also being joined with models or expert systems that predict pest and disease damage on crops, thereby allowing the impact of those important factors to be taken into account as well.

The limitation of requiring a user to interpret a model's results correctly is being overcome through the coupling of crop models with expert systems. Crop expert systems contain the reasoning abilities and knowledge of human agronomy experts. Such a system can run a crop model for a given situation and interpret the model's results for that situation.

All of these advancements in modeling have helped to bring crop models from research laboratories to growers' farms. The combining of expert systems with crop models has probably been the single greatest factor in making crop models usable as within-season management aids by growers. When used concurrently with a growing crop, an expert system can obtain information from the grower regarding actions performed on the crop and obtain daily weather data from a nearby weather station. With this information, it can run a

model for the crop in real-time and interpret the output. The expert system can then inform the grower of the present status of the crop and if the crop is experiencing any stress. Using long-term weather forecasts or historical means, the model could also be used to predict potential stresses for the week ahead, providing the grower with advance information for facing ongoing management decisions. Additionally, the expert system can use the model to predict the expected status of the crop at harvest, trying different management strategies and weather scenarios. The most profitable and efficient strategy would then be presented to the grower in much the same way as human crop consultants provide management recommendations to growers.

An example of this type of system is COMAX/Gossym, an expert system joined with a model for use with cotton crops (Lemmon, 1986). COMAX uses the Gossym cotton model to provide to the grower information on irrigation schedules, nitrogen requirements, and the crop maturity date. On-farm tests of this system have shown it's ability to assist a grower in improving crop yield and quality (Lemmon, 1986).

Systems such as COMAX/Gossym that fill the role of crop management tools are termed "decision support systems" (DSS), being computer programs that assist a grower in making profitable crop management decisions. Researchers in the Crop and Soil Science and Bioresource Engineering Departments at Oregon State University are in the process of

developing a DSS for growers in Oregon and elsewhere.

As shown by a simple diagram in Figure 1, the OSU-DSS will contain at least four main components: a user interface, an expert system, a crop model, and a weather data manager program used by the expert system to obtain actual and generated data for the crop model. Models of pest and disease damage could also be included in the DSS.

The user interface will allow the DSS to be an easy to operate, menu driven program, similar to the majority of commercially produced computer programs available today.

The expert system will contain the knowledge and inferencing capabilities to interpret the model(s) used and provide recommendations for the grower.

The crop model used in the DSS, as well as the expert system knowledge base, should remain independent of the rest of the DSS. This permits the basic DSS design to be used for the management of any crop for which a model and knowledge base are developed. Crop models used by the International Benchmark Sites Network for Agrotechnology Transfer (IBSNAT) will be used in the DSS. These models all use input files of a common format, allowing the same weather data, soil data, etc. to be used by any of these models. The CERES-Wheat model is one of the IBSNAT models and it will be the crop model used initially as the DSS is developed. CERES-Wheat has formerly been used as a research tool and will require some revision for use with the DSS.

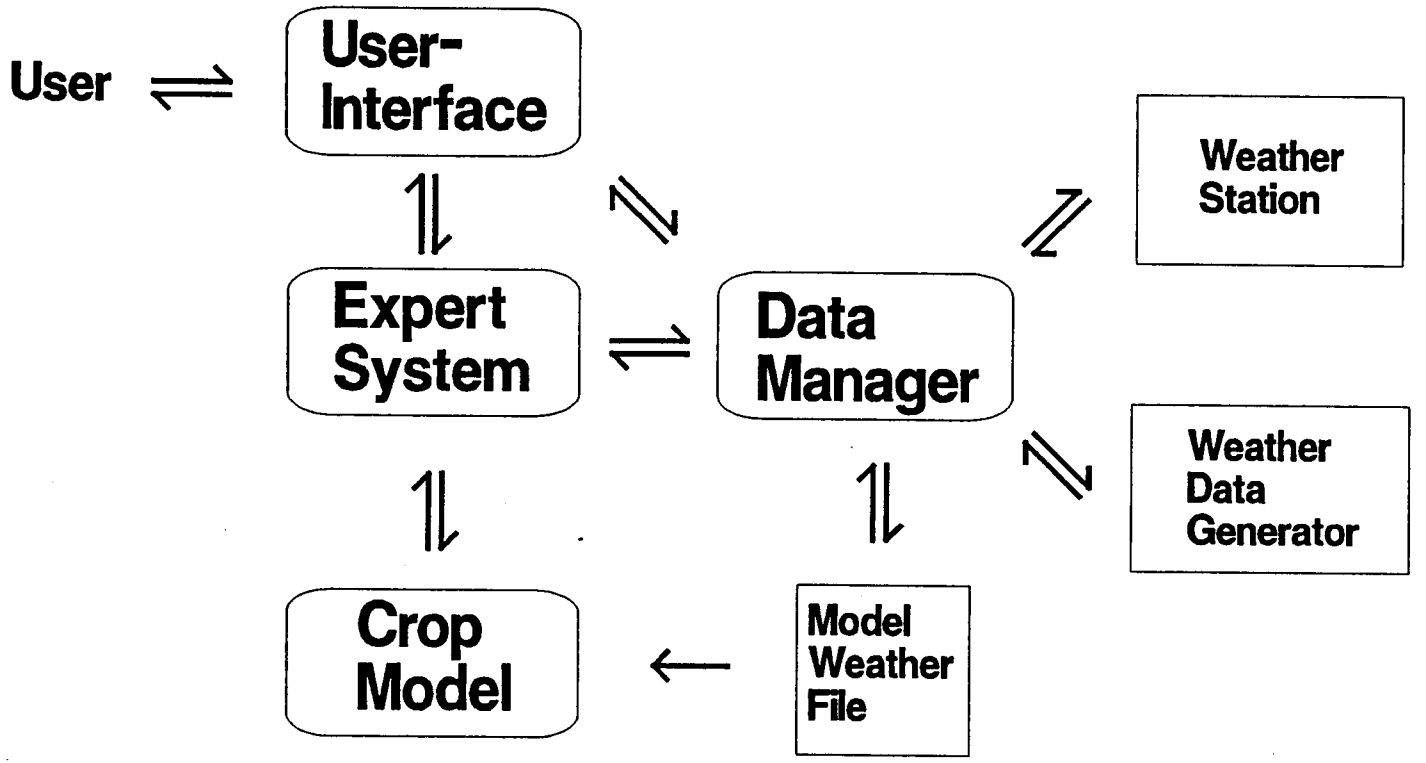


Figure 1. Some components of the proposed OSU Decision Support System.

The final section of the DSS is the weather data manager program. This program must provide the following functions for the DSS:

1) Retrieve weather data from a weather station. An automated weather station located on the farm where the DSS is used would provide an on-site source of current weather data for the crop model. This station should be linked by modem to the computer running the DSS so data from the station can be retrieved easily.

2) Run quality control procedures on collected data. Quality control (QC) of data from the station must be done to detect errant data due to instrument failure, communication problems, etc. The user should be able to view and, if necessary, correct any data that fail the QC procedures.

3) Place data in a file for use by the crop model. Daily data that the crop model requires (maximum and minimum temperatures, precipitation, and solar radiation) must then be placed in a file in a format readable by the model.

4) Provide for file editing. Because all errors cannot be detected and corrected by the QC procedures, the program must allow the user to edit the weather data file, as well as other files, such as the file containing the information used in the QC procedures.

5) Run a weather generator program. Generated weather

data are necessary for the model to predict the status of the crop at the future harvest. Weather generator programs have been developed to provide daily weather data for typical years at a given location. The data manager program must be able to call a weather generator, obtain the data for the days remaining until harvest, and place the data in a formatted file usable by the model. It would also be desirable to let the grower specify what the weather for the rest of the season might be like, such as cool and wet or warm and dry.

The objectives of this research were to obtain a weather station for use with the DSS, develop a set of QC procedures for data from the station, obtain a weather generator program for use with the DSS, and create the weather data manager program as described above.

LITERATURE REVIEW

Decision Support Systems

The "first attempt" at combining a crop model with an expert system to help growers optimize crop production was COMAX/Gossym, developed by the USDA at Mississippi State and Clemson Universities (Lemmon, 1986). Gossym is a simulation model of cotton and COMAX is the rule-based expert system that "operates Gossym the way a human expert would to determine three factors: irrigation schedules, nitrogen requirements, and the crop maturity date" (Lemmon, 1986). COMAX hypothesizes a management strategy for the grower at the beginning of the season, and as the hypothesized values of weather, irrigation, and fertilization are replaced by actual data, the strategy is updated and optimized (Whisler, et al., 1986). Hypothesized weather is in the form of normal, hot-dry, and cold-wet scenarios that are specific for the site where the system is used, and actual weather is collected daily from an on-site weather station (Lemmon, 1986). Initial tests of COMAX/Gossym were successful, with its recommendations resulting in increased yields (Lemmon, 1986). Development and testing is continuing with more features being added to the system, such as a new user-interface and ability to make decisions about defoliation (Ferguson, 1990).

Another type of DSS is CALEX/Cotton, an "integrated

expert decision support system" developed at the University of California-Davis (Plant, 1989a; Plant, et al., 1989). This "user-friendly" package is directed at growers, consultants, and even educational instructors (Plant, 1989a). CALEX, a general shell which provides the decision-making, is coupled with domain-specific modules for "agronomic management, pest management, economic management, and diagnosis" (Plant, 1989a). These modules are being developed for cotton and peaches. The system uses weather data (input directly by the user, downloaded from a weather database network, or taken from historical weather files), scouting data (insect counts, etc.), field characteristics, and activity data (fertilizer applications, etc.) in deciding field conditions and producing management schedules (Plant, 1989a). Any conflicts that arise in scheduled management activities are handled by part of the system called the "critic" that works to reschedule the actions (Plant, 1989b). Initial response to CALEX/Cotton and the lesser-developed CALEX/Peaches has been favorable and work is continuing each year to improve the system (Plant, 1989a; Plant, et al., 1989; Ferguson, 1990).

The International Benchmark Sites Network for Agrotechnology Transfer (IBSNAT) has developed and is continuing to work on a package entitled Decision Support System for Agrotechnology Transfer (DSSAT), which is not intended for growers but for scientists involved in the

various aspects of agrotechnology (Jones, et al., 1989). The program includes crop simulation models for wheat, maize, soybeans, and peanuts, along with a database to manage agronomic information and "an economic analysis program which helps the user determine effective management strategies" for a variety of factors, including cultivar, plant population, fertilizer, and irrigation (Tang, 1989). This system can only use historical or generated weather data and cannot be run simultaneously with a growing crop, but is designed to "predict the outcome of experiments never tried before, or test the desirability of a new cultivar, product, or management technique" that could eventually be adopted by farmers (Tang, 1989).

CERES-Wheat Simulation Model

The CERES-Wheat model was developed by the USDA Crop Systems Evaluation Unit at Temple, Texas and "was designed to be a decisionmaking tool for farmers, researchers, and governments" (Ritchie, et al., 1985). It is a daily-incrementing model providing yield estimation to users, along with estimation of such processes as dry-matter partitioning, phasic development, and impact of soil water and/or nitrogen deficit on growth (Ritchie, 1985). Inputs the model uses (soil, weather, genetic and management information) are readily obtainable, the model is written in the widely familiar FORTRAN language, and the time required

to execute the model is minimal. These are all qualities of CERES-Wheat that make it a "user-oriented" model (Ritchie, 1985). Some factors not taken into account in the model are such things as insects, weeds, and diseases. However, pest models can be linked to the wheat model to include these factors (Ritchie, et al., 1985). The model has "widespread applicability in diverse environments" (Godwin and Vlek, 1985) and because of this, it was chosen by IBSNAT for their purpose of sharing technology among countries (Uehara, 1985).

Quality Control of Weather Data

Data collected from the on-site automated weather station must be subjected to certain quality control procedures before being used by the model. Sources of errors in the data are, among other things, instruments out of calibration, malfunctioning instruments, data transmission, and various actions of animals affecting the instruments (Ashcroft, et al., 1990a). Although the "dynamic nature of weather makes it difficult to use strict, fixed criteria to quality control climatic data", certain checks can be implemented to detect obvious or possible errors (Ashcroft, et al., 1990b).

Reinke and Hollinger (1990) describe the quality control procedures used on hourly and daily data collected from automated weather stations in the Illinois Climate

Network. Four types of computerized checks (listed below) are performed, and data not passing any one of these checks are flagged for manual review.

1) Extreme value checks make sure each hourly and daily variable is within allowable limits. For example, daily maximum air temperature must be within -30 to 40 degrees Celcius. Ashcroft, et al. (1990b) make this check more strict by having separate limits for each month, so a January daily maximum air temperature is checked against the high and low values for maximum temperatures in January. This same type of check can be performed on all other variables such as precipitation and solar radiation.

2) Comparison checks are a second set of control procedures performed. These tests make sure the data do not contain obvious errors, such as positive nighttime solar radiation readings or maximum temperature less than minimum temperature.

3) Missing/duplicate checks make sure there is one and only one hourly and daily data record for each hour and day.

4) Time consistency checks flag certain variables that do not change or change by more than a reasonable amount within a determined period of time. For example, hourly air temperature readings that are the same for three hours or more cause that data to be

flagged. This check is very useful for identifying malfunctioning instruments or power failures.

Weather Data Generation

Although several programs have been developed for generating daily weather data (Nicks and Harp, 1980; Jones, et al., 1972; Samani and Hargreaves, 1987; Richardson and Wright, 1984; Zuzel and Karow, 1988) three are readily available for use.

WGEN (Richardson and Wright, 1984) generates daily sequences of precipitation, solar radiation, and maximum and minimum temperature for any number of years. This program uses several parameters which are established for weather stations in the United States. Using the WGEN PAR program, these parameters can also be produced from historical weather data for any specific site, if desired. These parameters are first used to generate precipitation (if any) for a given day, then solar radiation and maximum and minimum temperatures are generated based upon the wetness or dryness of the day. WGEN is "designed to preserve the dependence in time, the correlation between variables, and the seasonal characteristics in actual weather data for the location."

WEATHERWIZARD (Zuzel and Karow, 1988), based on the WGEN model, generates ten different daily sequences of precipitation, solar radiation, and maximum and minimum

temperature. These sequences are generated using either data and parameters from nineteen weather stations or, for a site away from one of these stations, using a triangulation process. This method of generation limits the program's use to north central Oregon, where it was created.

Samani and Hargreaves (1987) describe the WMAKER program developed at Utah State University by A.A. Keller. This weather generator uses historical monthly means in generating weather sequences. Daily evapotranspiration values are generated and randomized. Then, from these values, daily temperature and solar radiation are calculated. Rainfall is then randomly generated "to coincide with days of low solar radiation (cloudy days)." The program is designed for "world-wide" use.

MATERIALS AND METHODS

Weather Station

For a weather station to be used with the DSS, it must contain at least the necessary sensors for accurately measuring daily total precipitation, daily total solar radiation and daily maximum and minimum temperatures. It must have its own power source, enabling it to be placed in any open area on a grower's farm, where power lines may not run. Finally, the station must be able to store the collected data for a limited period of time and permit the data to be retrieved by means of a personal computer.

A station meeting all of the above requirements was obtained from Campbell Scientific, Inc., Logan, Utah. The separate and total costs of the Campbell 012 station are given in Table 1. A diagram of the station and its sensors is given in Figure 2. The datalogger inside the 012 station is programmed to read each sensor every ten seconds and provide output of hourly summaries, daily summaries, and precipitation occurrences as listed in Table 2. The program is contained in "programmable read only memory" so the program is executed as soon as power is connected to the datalogger (or as soon as power is restored if a power outage occurred). This feature is desirable because programming the datalogger is a complex process which a grower should not have to learn. However, this resident

Table 1. Costs of Campbell 012 station and related materials as of December 1989 (shipping not included).

Model #	Description	Price
012	Weather station Includes CR10 datalogger and the following sensors: temperature and RH (4472), wind direction (4461), wind speed (4462), pyranometer (4813), rain gauge (4464).	$3,320.00
PC208	Datalogger support software	200.00
SRM-6A	RAD short haul modem (2 @ $100.00)	200.00
5563	4-wire surge protector	15.00
SC932	9-pin to RS232 DCE interface	130.00
MSX10R	10 Watt solar panel, regulator, and mounts	230.00
5240	Power Sonic 12V 6.5 Ahr battery	30.00
		$4,125.00

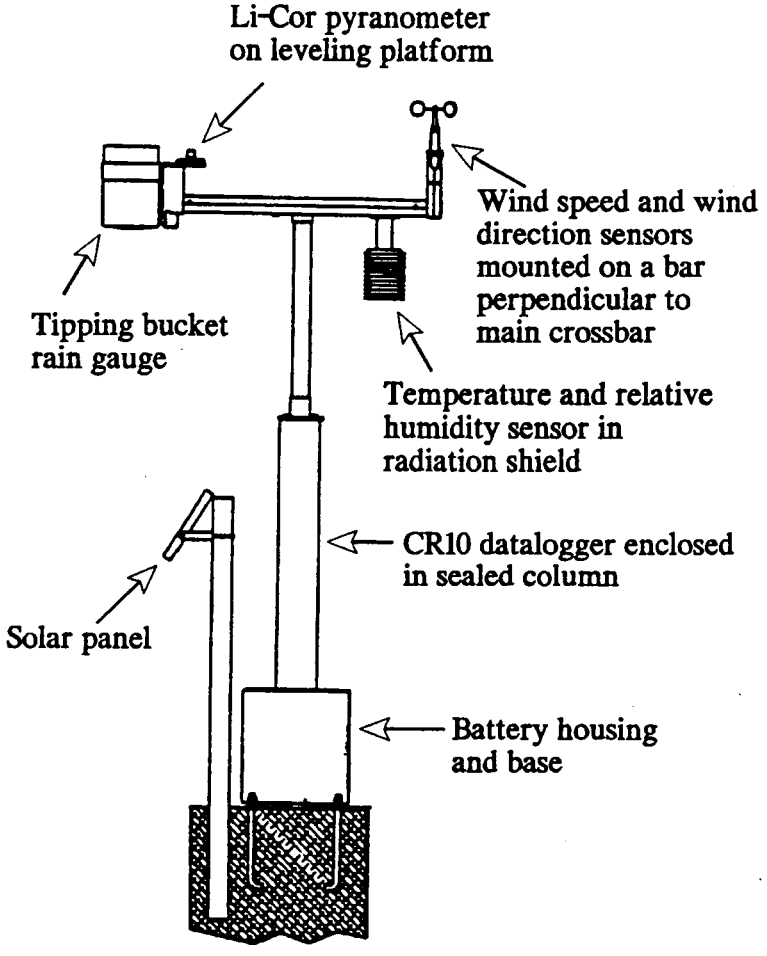


Figure 2. Front view of Campbell 012 weather station with solar panel (adapted from Campbell Scientific, Inc., 1989).

Table 2. Output from the CSI 012 weather station datalogger using the standard program.

Hourly Output	Units
ID of dataset	3-digit number
Julian date	day number
Hour, minute	24-hour format
Ave. temperature	degrees F
Instantaneous relative humidity	%RH
Ave. solar radiation	kW/sq. meter
Ave. wind speed	mph
Weighted ave. wind direction	degrees
Std. dev. of wind direction	degrees

Daily Output	Units
ID of dataset	3-digit number
Julian date	day number
Hour, minute	24-hour format
Ave. temperature	degrees F
Max. temperature	degrees F
Min. temperature	degrees F
Max. relative humidity	%RH
Min. relative humidity	%RH
Ave. solar radiation	kW/sq. meter
Max. wind speed	mph
Ave. wind speed	mph
Total precipitation	inches
Datalogger moisture	index
Battery voltage	volts
Datalogger max. temperature	degrees C
Datalogger min. temperature	degrees C

Conditional Output	Units
ID of dataset	3-digit number
Hour, minute	24-hour format
Precipitation amount	inches

program is undesirable in two ways: an average daily solar radiation reading is provided instead of the necessary daily total solar radiation and the measurements are in English units instead of the SI units used by the crop models. To handle these differences, appropriate conversions are performed on the data after collection from the datalogger.

The datalogger is capable of communicating with an IBM-PC or compatible computer via the Campbell PC-208 software and a set of modems. Communication can occur over telephone lines with phone modems, radio frequencies with RF modems, or direct wire communication cable with short-haul modems. The short-haul modems were used for this research, allowing inexpensive communication up to distances of 8 km, if necessary.

Power for the station can come from a 12 volt DC battery or from 110 volt AC lines via a transformer. For placement in a grower's field, the option of having a battery charged by a solar panel seemed best. This option allows the station to be situated away from AC power sources and eliminates the need of replacing drained batteries.

The 012 station comes designed to set on a permanent concrete base. However, to make this station more portable, a different base was designed and constructed, as shown in Figure 3.

The 012 station was initially assembled in a lab and linked to a personal computer. After determining that it

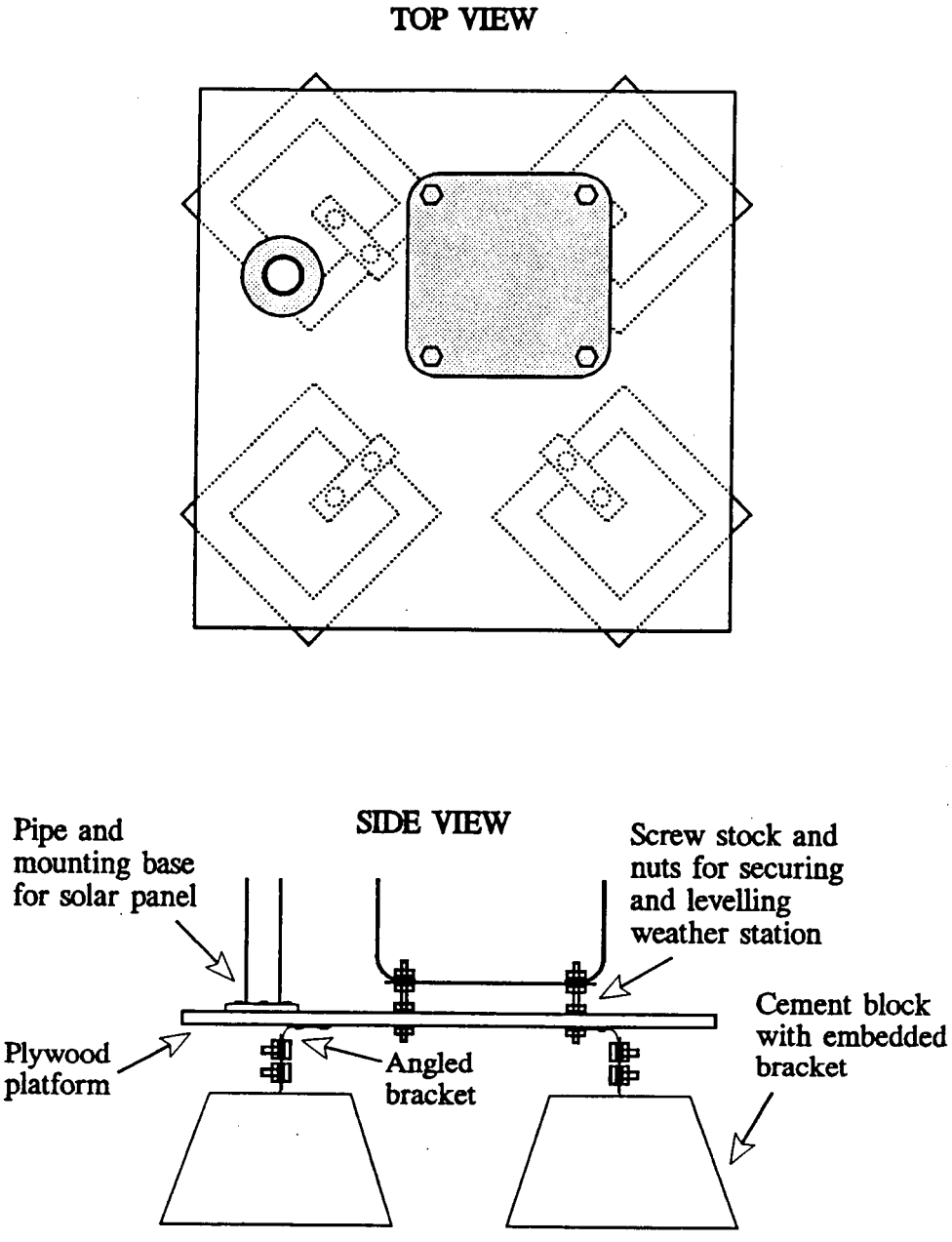


Figure 3. Portable base constructed for 012 weather station and solar panel.

was operating correctly, the station was placed at Hyslop Field Laboratory, Corvallis, Oregon, in close proximity to the Cooperative weather station located there, for the purpose of comparing the Campbell station's data with another station's. The 012 station was not linked with a computer while located at Hyslop. However, a portable computer was taken to the site approximately once a month and connected to the station via the short-haul modems to retrieve the collected weather data. The station's sensors were also checked when data was retrieved to ensure that they were clean and working properly.

The existing Cooperative station at Hyslop, from which data is read manually, is being replaced by an automated station, which is in the AGRIMET weather station network. The AGRIMET station began operating just prior to the time the Campbell station was placed at Hyslop, but the Cooperative station will continue in use until the AGRIMET station has been proven to operate satisfactorily. Therefore, it was possible to compare data from the Campbell station with data from these two other weather stations.

For five months the Campbell station was located about one kilometer from the other two weather stations, but was then moved immediately adjacent to the other stations. Data from the Cooperative and AGRIMET stations were obtained for the ten month period (May to February) when the Campbell station was at Hyslop.

The values used in the data comparison were daily maximum and minimum temperature, daily total precipitation, and daily total solar radiation (the four weather inputs used by IBSNAT models). From the daily values for these parameters, monthly totals for precipitation and monthly means for maximum and minimum temperature and daily total solar radiation were calculated. The monthly values from the Cooperative and AGRIMET stations were then averaged. A t-test was performed on the differences between these averaged values and the corresponding monthly values from the Campbell station. (Note: Data from all three weather stations were received in non-SI units. To reduce errors and simplify calculations, the data were not converted to SI units.)

Quality Control of Data

The quality control procedures used on the data from the automated weather station were limited by the assumptions that the station will not be near any official weather stations and, if it is, that the official data will not be readily available. This meant that the QC procedures could not involve comparison of the automated station's data with data from any other stations in the area (a common practice in QC of weather data networks). Also, it is assumed that the grower will want to spend a minimal amount of time, if any, checking the weather data, so the

procedures should be as automated as possible.

The QC procedures used in this project were adapted from Reinke and Hollinger (1990). They gave several types of automated checks which can be used without requiring another station for comparison. A short description of each type of test follows.

Extreme Value Tests. A value is rejected if it is not within a range normally expected for that value.

Constant Value Tests. A value is rejected if it is constant for three hours or more (hourly data) or two days or more (daily data).

Comparison Tests. A value is compared with another value to determine if the relationship between the two is unreasonable. For example, a daily maximum temperature value would be rejected if it is less than the corresponding daily minimum temperature value.

Excessive Change Tests. A value is compared with the corresponding value for the previous hour or day to determine if there has been an increase or decrease of more than a certain amount.

Duplicate/Missing Values Test. This check ensures that each hourly or daily summary contains one and only one value for each weather characteristic measured.

Using these types of tests, a set of QC procedures was developed for the data collected by the 012 station. These procedures were put into the data manager program to be

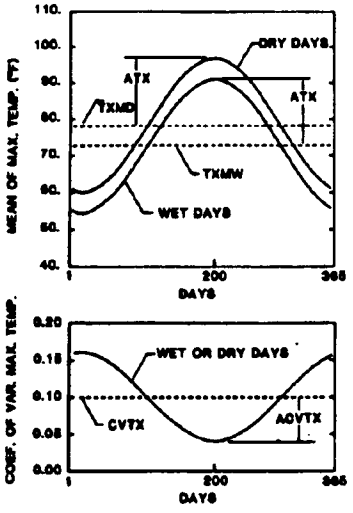
performed automatically when any data is retrieved from the weather station. To test each procedure, data files were obtained from the 012 station and were altered to include values which would fail a given test. The QC procedures were then run on the altered data files to ensure that the errant values were identified. Additionally, several months of unaltered data from the station were run through the QC procedures to determine if any errant data could be detected.

Weather Data Generator

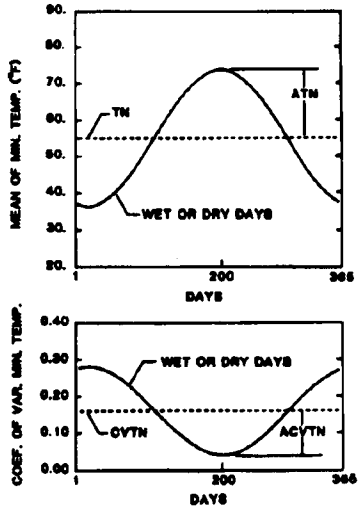
A weather generator to provide weather data from the present day to the end of the growing season was selected from the ones presented in the literature. WGEN was chosen for this project because it can generate data for any location in the contiguous United States. Also, it does not require any historical data for a location, which a grower might not have access to.

To run WGEN, several input parameters are required. These parameters are listed in tables and charts in Richardson and Wright (1984). Each parameter is listed below, followed by a short description. Figure 4, adapted from Richardson and Wright (1984), shows graphically the parameters used for generating maximum temperature, minimum temperature, and solar radiation.

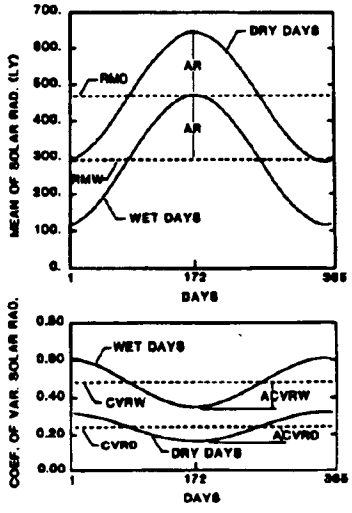
1) Years of data to generate: the number of years of



A, maximum temperature



B, minimum temperature



C, solar radiation

Figure 4. Description of parameters used by WGEN for generating maximum temperature (A), minimum temperature (B), and solar radiation (C); adapted from Richardson and Wright (1984).

generated data desired.

2) Generation option code: set to '1' if precipitation is to be generated along with maximum and minimum temperature and solar radiation; set to '2' if actual precipitation data will be used to generate maximum and minimum temperature and solar radiation.

3) Latitude: latitude of the location for which data are being generated.

4) Temperature correction code: set to '0' if no temperature correction is used; set to '1' if using monthly values of actual mean temperature for temperature correction; set to '2' if using monthly values of actual mean maximum and minimum temperatures for temperature correction.

(The following four parameters are used for rainfall.)

5) Monthly probability of wet day, given wet day on previous day: the probability of a wet day occurring given that a wet day occurred the previous day; one probability value for each month.

6) Monthly probability of wet day, given dry day on previous day: the probability of a wet day occurring given that a dry day occurred the previous day; one probability value for each month.

7) Monthly values of gamma distribution shape parameter (alpha): the shape parameter of the gamma distribution used to describe the distribution of rainfall amounts;

one value for each month.

8) Monthly values of scale distribution shape parameter (beta): the scale parameter of the gamma distribution used to describe the distribution of rainfall amounts; one value for each month.

(The following five parameters are used for max. temp.)

9) Mean of max. temperature for dry days (TXMD in Figure 4).

10) Amplitude of mean of max. temperature for wet or dry days (ATX in Figure 4).

11) Mean of the coefficient of variation for max. temperature for wet or dry days (CVTX in Figure 4).

12) Amplitude of the mean of the coefficient of variation for max. temperature for wet or dry days (ACVTX in Figure 4).

13) Mean of max. temperature for wet days (TXMW in Figure 4).

(The following five parameters are used for min. temp.)

14) Mean of min. temperature for wet or dry days (TN in Figure 4).

15) Amplitude of mean of min. temperature for wet or dry days (ATN in Figure 4).

16) Mean of the coefficient of variation for min. temperature for wet or dry days (CVTN in Figure 4).

17) Amplitude of the mean of the coefficient of variation for min. temperature for wet or dry days

(ACVTN in Figure 4).

(The following three parameters are used for solar rad.)

18) Mean of solar radiation for dry days (RMD in Figure 4).

19) Amplitude of mean of solar radiation for wet or dry days (AR in Figure 4).

20) Mean of solar radiation for wet days (RMW in Figure 4).

(Parameters labeled in Figure 4 but not listed above are not location dependent.)

The WGEN program was written in the FORTRAN programming language. It was decided that the program should be rewritten in the C language to give it greater readability from a programmer's standpoint. The computer and the version of C used for the rewrite are the same used for the creation of the data manager program, described later.

Following the initial rewrite, results from a run of the C version were compared with results from the FORTRAN version, using the same input parameter file. Errors in the C version were corrected until results from the two versions were identical on all accounts. Modifications were then made to the C version to: redirect output from the screen to disk files; have it return a value to the program which called it, signifying that WGEN ran successfully or that an error occurred; and have it produce a different set of weather data each time it is run.

A parameter file for WGEN was established for Corvallis, Oregon using the appropriate tables and charts in Richardson and Wright (1984). The parameters in this file are given in Table 3. WGEN was run with this parameter file to produce 30 years of data. The 30-year monthly means from this run were compared with 30-year historical monthly means for Corvallis. Differences between the WGEN means and the corresponding historical means were calculated and a t-test was performed on these differences.

Data Manager Program

The largest portion of this research was the development of the weather data manager program described previously. The C programming language was chosen for writing the manager. Microsoft QuickC version 2.0 was used on an IBM-PC compatible computer with an 80286 microprocessor and an 80287 math coprocessor.

The program was written with the intent of keeping different functions of the program in separate modules, allowing for easier modification of the program in the future. Also, it was determined that little time should be spent on the program's user interface. All necessary data should be displayed and input required from the user should be retrieved in a "user-friendly" manner. However, the format of the screen displays in the program should be kept simple. The display format will be finalized when the user-

Table 3. Input information for WGEN for the Corvallis, Oregon area.

Comments:	[WEATHER INFO FOR WGEN FROM TABLES FOR CORVALLIS, OREGON]
Years of data to generate:	30
Generation option code:	1
Latitude:	44.0
Temp. correction option code:	0
Prec. correction option code:	0
Probability of wet given wet:	0.791 0.728 0.750 0.638 0.611 0.555 0.404 0.494 0.507 0.659 0.776 0.755
Probability of wet given dry:	0.411 0.341 0.293 0.304 0.215 0.151 0.045 0.086 0.148 0.233 0.339 0.427
Alpha:	0.866 0.763 0.964 0.867 0.998 0.776 0.826 0.829 0.722 0.866 0.833 0.827
Beta:	0.435 0.380 0.270 0.198 0.172 0.221 0.148 0.157 0.296 0.337 0.380 0.434
Coefs. of max. temp. for dry days:	65.00 24.00 0.14 -0.06
Coef. of max. temp. for wet days:	59.00
Coefs. of min. temp.:	40.00 13.00 0.17 -0.08
Coefs. of radiation for dry days:	420.00 302.00
Coef. of radiation for wet days:	267.00

interface is written for the entire DSS.

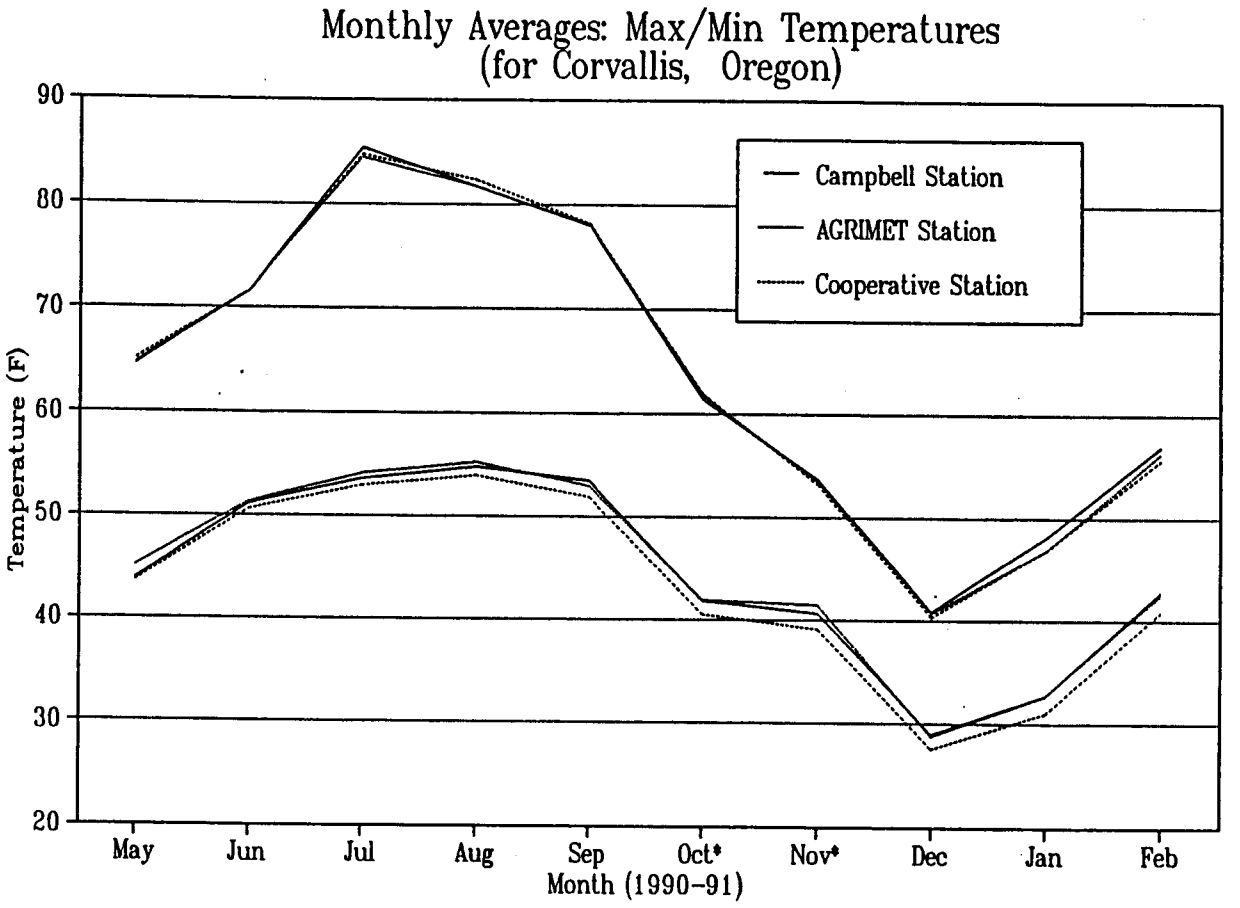
Approximately one month was spent learning the C programming language. The manager program was written, tested, evaluated, and modified over a period of six months.

RESULTS AND DISCUSSION

Weather Station

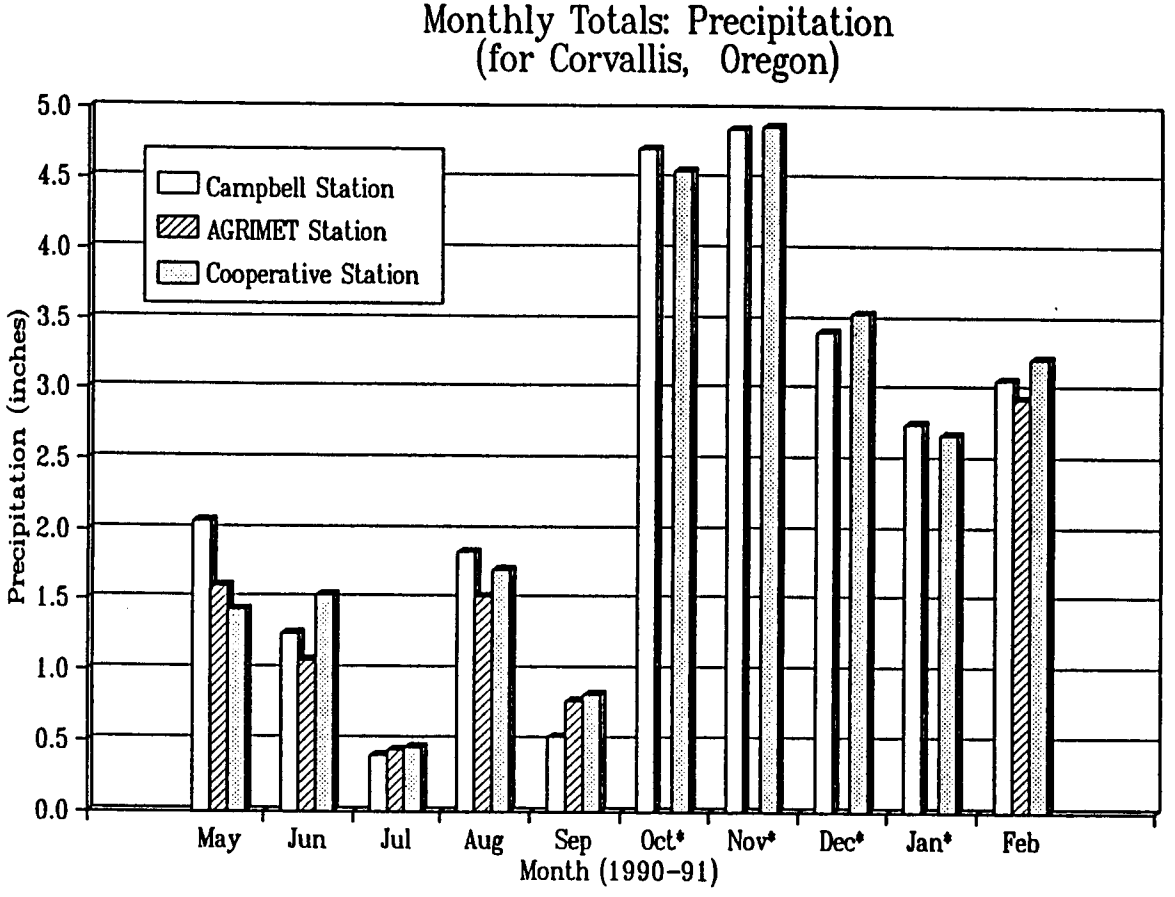
The Campbell station performed well during its ten months of operation. Regular checking of the station's sensors proved essential in ensuring their proper operation. The surface of the pyranometer required frequent cleaning (approximately once a month) to remove dust and other material which collected on it. The collection funnel of the rain gauge required checking to make sure it was unobstructed. The most significant problem occurred in September when the collection funnel became plugged with remains from bird feeding and excretion. The longest the plug could have been in the funnel was three weeks (the time since the previous data collection and check). The plug apparently affected precipitation readings, as can be seen from September's precipitation data, shown later in Figure 6. Brief weekly checks of the station's sensors would help prevent such problems from affecting data.

The quality of the data from the Campbell weather station is, for the most part, good. Graphs of the data from the three stations are shown in Figures 5-7. Several gaps in the data from the new AGRIMET station, caused by various problems with the station's sensors and with data transmission, were unfortunate and decreased the number of months for which data was compared. For the months with



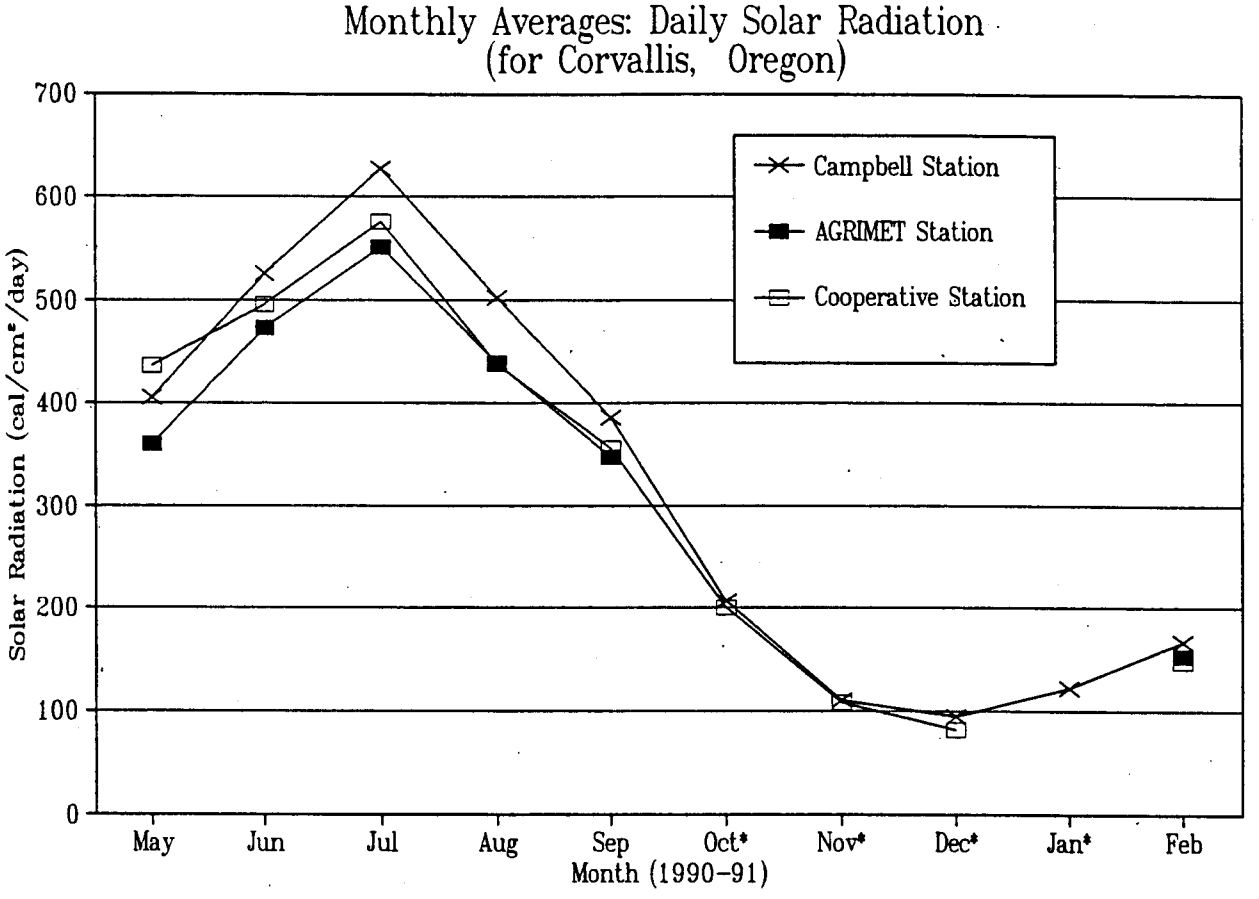
*AGRIMET Station missing data for Oct 30-31; Nov 1.

Figure 5. Monthly averages of maximum and minimum temperatures for Corvallis, Oregon from the Campbell, AGRIMET, and Cooperative stations.



*AGRIMET Station missing data for Oct 30-31; Nov 1, 15-30; Dec 1-31; Jan 1-14.

Figure 6. Monthly totals of precipitation for Corvallis, Oregon from the Campbell, AGRIMET, and Cooperative stations.



*AGRIMET Station missing data for Oct 30-31; Nov 1, 15-30; Dec 1-31; Jan 1-18; Cooperative Station missing Jan 1-31.

Figure 7. Monthly averages of daily total solar radiation for Corvallis, Oregon from the Campbell, AGRIMET, and Cooperative stations.

complete data from all three stations, some trends can be seen. Maximum temperature data agreed well between all three stations. Minimum temperature data agreed well between the Campbell and AGRIMET stations, with the Cooperative station data being consistently slightly lower. For precipitation, no station measured consistently higher or lower. The graph of daily solar radiation shows the most obvious trend of the Campbell station producing data which is considerably higher than the other two stations.

Tables 4-7 list, for each weather parameter compared, the monthly values from the Campbell station, the averaged monthly values from the Cooperative and AGRIMET stations, differences between the values for months which had complete weather data from all three stations, and the P-value resulting from a t-test on these differences.

At the .05 significance level, there were no significant differences between the data from the Campbell station and the averaged data from the other two stations for monthly mean maximum and minimum temperature and monthly total precipitation. However, the monthly means of daily total solar radiation did differ significantly.

It was possible to further compare the solar radiation data from the Campbell station with data from an Eppley thermopile pyranometer. The Eppley was being used for other research, but was located immediately adjacent to the Campbell station for ten days measuring daily total solar

Table 4. Monthly means of maximum temperature from Campbell station, averaged monthly means from AGRIMET and Cooperative stations, differences between the means, and P-value from t-test on the differences.

MAX. TEMPERATURE	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Campbell station	64.64	71.57	85.30	81.66	77.88	61.38	53.60	40.69	48.01	56.89
Averaged stations	64.85	71.58	84.54	81.92	78.14	61.79	53.24	40.41	46.73	55.91
Difference	-0.21	-0.01	0.76	-0.26	-0.25			0.28	1.28	0.98
P-value	0.1822									

Table 5. Monthly means of minimum temperature from Campbell station, averaged monthly means from AGRIMET and Cooperative stations, differences between the means, and P-value from t-test on the differences.

MIN. TEMPERATURE	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Campbell station	43.77	51.13	53.54	54.71	53.43	41.70	40.61	28.96	32.61	42.69
Averaged stations	44.26	51.00	53.45	54.53	52.42	41.16	40.23	28.07	31.74	41.54
Difference	-0.49	0.14	0.08	0.17	1.00			0.89	0.87	1.14
P-value	0.0523									

Table 6. Monthly totals of precipitation from Campbell station, averaged monthly totals from AGRIMET and Cooperative stations, differences between the totals, and P-value from t-test on the differences.

PRECIPITATION	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Campbell station	2.06	1.26	0.39	1.85	0.53	4.71	4.85	3.41	2.76	3.07
Averaged stations	1.52	1.30	0.44	1.62	0.81	3.77	3.32	1.77	1.60	3.08
Difference	0.54	-0.04	-0.05	0.23	-0.28					-0.01
P-value	0.5986									

Table 7. Monthly means of daily total solar radiation from Campbell station, averaged monthly means from AGRIMET and Cooperative stations, differences between the means, and P-value from t-test on the differences.

SOLAR RADIATION	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Campbell station	404.91	525.96	626.82	501.94	386.68	206.63	110.82	95.26	122.30	166.82
Averaged stations	398.95	483.95	563.86	438.35	352.47	196.29	106.77	41.00	76.99	150.48
Difference	5.96	42.00	62.96	63.59	34.21					16.35
P-value	0.0116									

radiation. Again, the Campbell station measured consistently higher than the Eppley, as is shown in Figure 8, a graph of hourly total solar radiation on a clear day in July.

It was apparent that the Campbell station was not calibrated properly for solar radiation. The standard program contained in the station's datalogger has a multiplier of 0.2 Kw m^{-2} mV^{-1} to convert the Mv reading from the Li-Cor pyranometer to an instantaneous measurement of solar radiation in Kw m^{-2}. It was thought that this standard multiplier may not be proper for the pyranometer which came with this station.

A call was made to Campbell Scientific, Inc. regarding this matter. A Campbell representative stated that this type of problem had been encountered before but not to this degree. He agreed that the multiplier in the program is not adequate for this particular pyranometer sensor. He suggested that the sensor could be returned to Campbell for recalibration or the data from the Eppley pyranometer could be used to correct the multiplier. The latter option was chosen. Ratios of the Campbell data and Eppley data were calculated and a correction factor was obtained. Monthly means of total daily solar radiation with the Campbell data corrected are shown in Figure 9.

This correction factor could be included in an updated program which would be downloaded to the weather station's

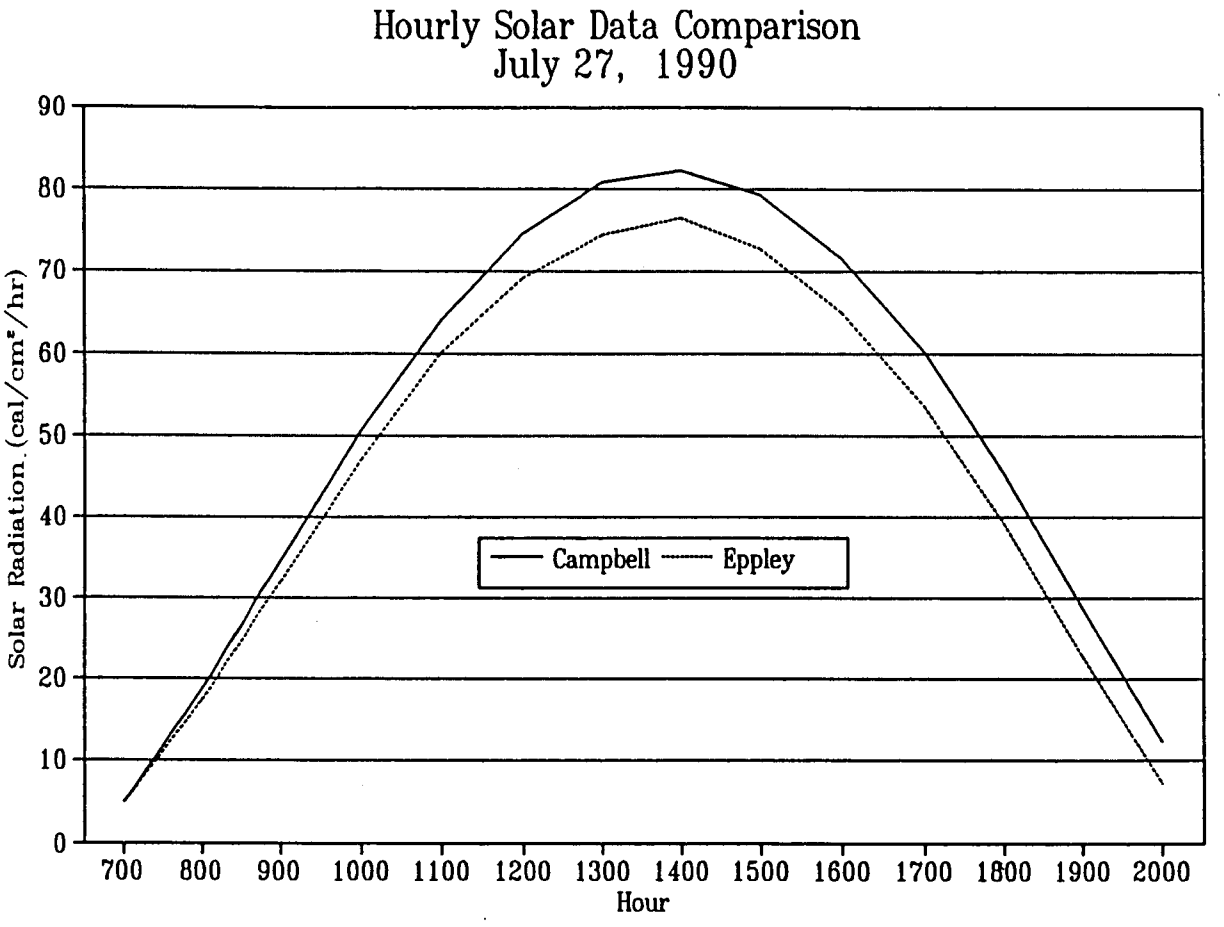
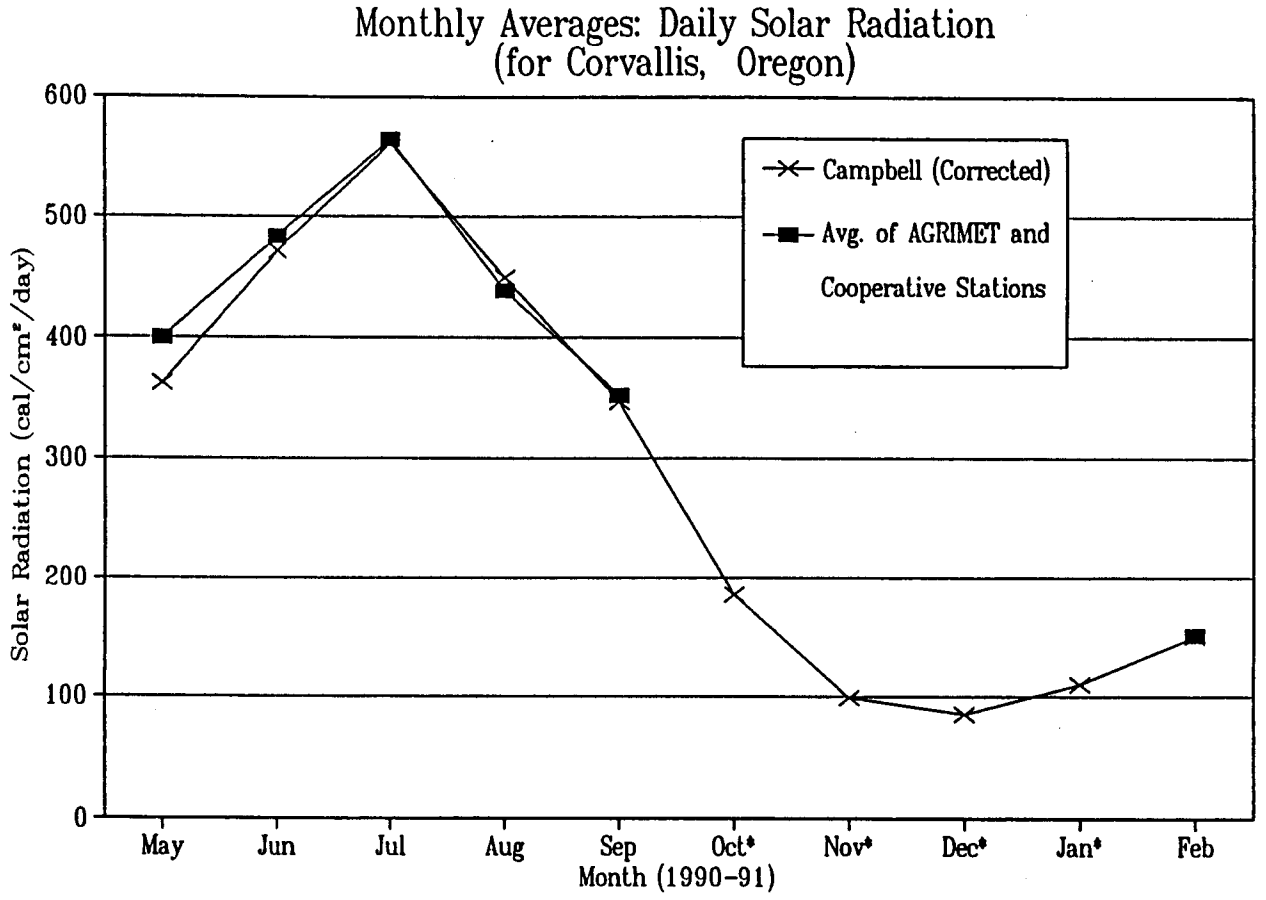


Figure 8. Hourly total solar radiation for Corvallis, Oregon from the Campbell station and Eppley pyranometer.



*AGRIMET Station missing data for Oct 30-31; Nov 1, 15-30; Dec 1-31; Jan 1-18; Cooperative Station missing Jan 1-31.

Figure 9. Monthly averages of daily total solar radiation for Corvallis, Oregon with the Campbell data corrected.

datalogger. However, as stated before, programming the datalogger should be avoided. The other option is to correct the data once it is retrieved from the datalogger. This could be accomplished by having the data manager program store user-supplied correction factors for all of the sensors. (Sensors which need no correction factor would have a factor of one). The factors would then be multiplied with the corresponding data value after it is read from the raw data file.

An important principle was learned from this problem. It cannot always be assumed that a weather station sensor (or any other instrument) is properly calibrated when purchased new from a manufacturer. A "standard" of some type should always be available for comparison. Although this comparison can usually be relatively easy at a university, it can be extremely difficult for a grower to perform on his or her farm. It may be beneficial for a weather station to be tested at the university prior to its placement at a grower's farm and subsequent use with the DSS.

Quality Control of Data

One or more of the types of QC tests described previously were used on most of the values output by the Campbell 012 station. The values tested and the types of tests used on each value are listed in Table 8.

Table 8. Types of quality control tests used on selected data from the Campbell 012 weather station.

Hourly Values	Tests
Julian date	Extreme
Hour	Missing/Duplicate
Ave. temperature	Extreme, Constant
Inst. relative humidity	Extreme, Constant
Ave. solar radiation	Comparison

Daily Values	Tests
Julian date	Extreme, Missing/Duplicate
Max. temperature	Extreme, Constant, Excessive Change, Comparison
Min. temperature	Extreme, Constant, Excessive Change
Max. relative humidity	Extreme, Constant, Comparison
Min. relative humidity	Extreme, Constant
Total solar radiation	Extreme
Max. wind speed	Constant
Total precipitation	Extreme
Datalogger moisture	Extreme
Battery voltage	Extreme

Most tests are used as described previously, but some require further explanation. The duplicate/missing value test on the hour and Julian date is used to ensure that no hourly or daily data sets are missing. For the comparison test on hourly solar radiation, the hourly solar radiation reading is compared with the hour of the reading to make sure solar radiation is not positive at night. The comparison tests on the daily maximums of temperature and relative humidity involve comparing these maximums with the corresponding minimum to make sure a maximum is not less than a minimum.

For the values of total solar radiation, precipitation, and maximum and minimum temperatures, the extreme value tests are made more strict by using ranges which vary month-by-month. These ranges are best if obtained from monthly historical maximums and minimums for the area, but estimated extremes for each month would also work well. The extreme value checks on datalogger moisture and battery voltage are used to ensure proper operation of the weather station. The ranges for these tests are given in the weather station user's manual.

This set of QC procedures requires two files of data. The first file contains value ranges for the extreme value tests. Having these ranges in a file allows them to be specific for the area in which the station is located and also allows them to be easily entered and updated by the

grower.

The second file contains the previous two hourly data sets and the previous one daily data set from the weather station. These data sets are used in the constant value tests and duplicate/missing value tests. This file is automatically updated after a data set is run through the QC procedures.

A file of extreme values was developed for the Corvallis area based on long-term historical maximums and minimums. Using these values in the QC procedures, a test set of weather data files, into which different values had been manually introduced, were run through QC. All of the procedures worked as desired, detecting all of the introduced errant values. Additionally, several months of unaltered data from the 012 station were run through the procedures. The extreme value tests detected some data which was out of the given ranges and the constant value tests detected values (mostly relative humidity values) which were the same over three to four hours or two to three days. All of these values which failed QC were apparently legitimate values, except for some solar radiation values for reasons discussed earlier. Two conclusions were drawn from these results. First, the data from the weather station was again shown to be reasonable and reliable. Second, values which fail QC may be legitimate values; therefore, procedures are needed which allow the grower to

make quick and easy choices regarding what to do with these values. The procedures developed for handling the values which fail QC are described later.

Weather Data Generator

WGEN proved to be an easy-to-use, relatively fast program. To generate data for a year took approximately eight to nine seconds when WGEN was run on an IBM-AT computer. However, the data produced from WGEN using the input parameters for the Corvallis, Oregon area did not compare well with historical data for Corvallis. Figures 10-12 show the comparisons of the WGEN 30-year means and historical 30-year means. The graph of maximum and minimum temperature shows that the curve made by the WGEN data had too high of an amplitude, so temperatures are either too high or too low when compared with the historical means. Precipitation data compared well, which reflects the main strength of WGEN for producing precipitation amounts for an area. However, solar radiation data from WGEN again produced a curve with too high of an amplitude in relation to historical means.

The sets of data and the results of t-tests on the differences are given in Tables 9-12. Although the t-tests indicate significant differences ($p = .05$) in only the solar radiation data, it is clear from the graphs that the temperature data have important, but not consistent,

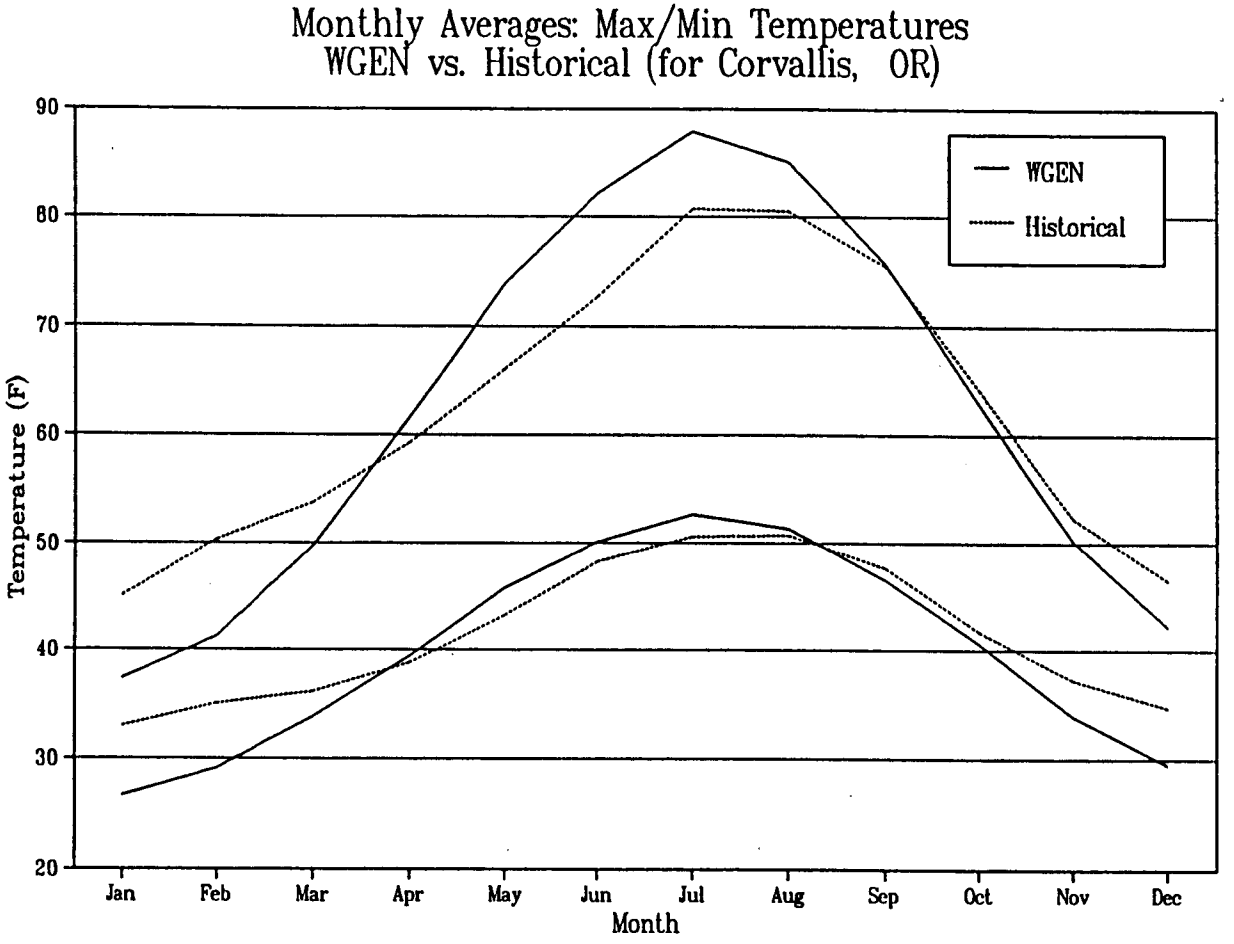


Figure 10. Monthly averages of maximum and minimum temperatures for Corvallis, Oregon from WGEN and historical records.

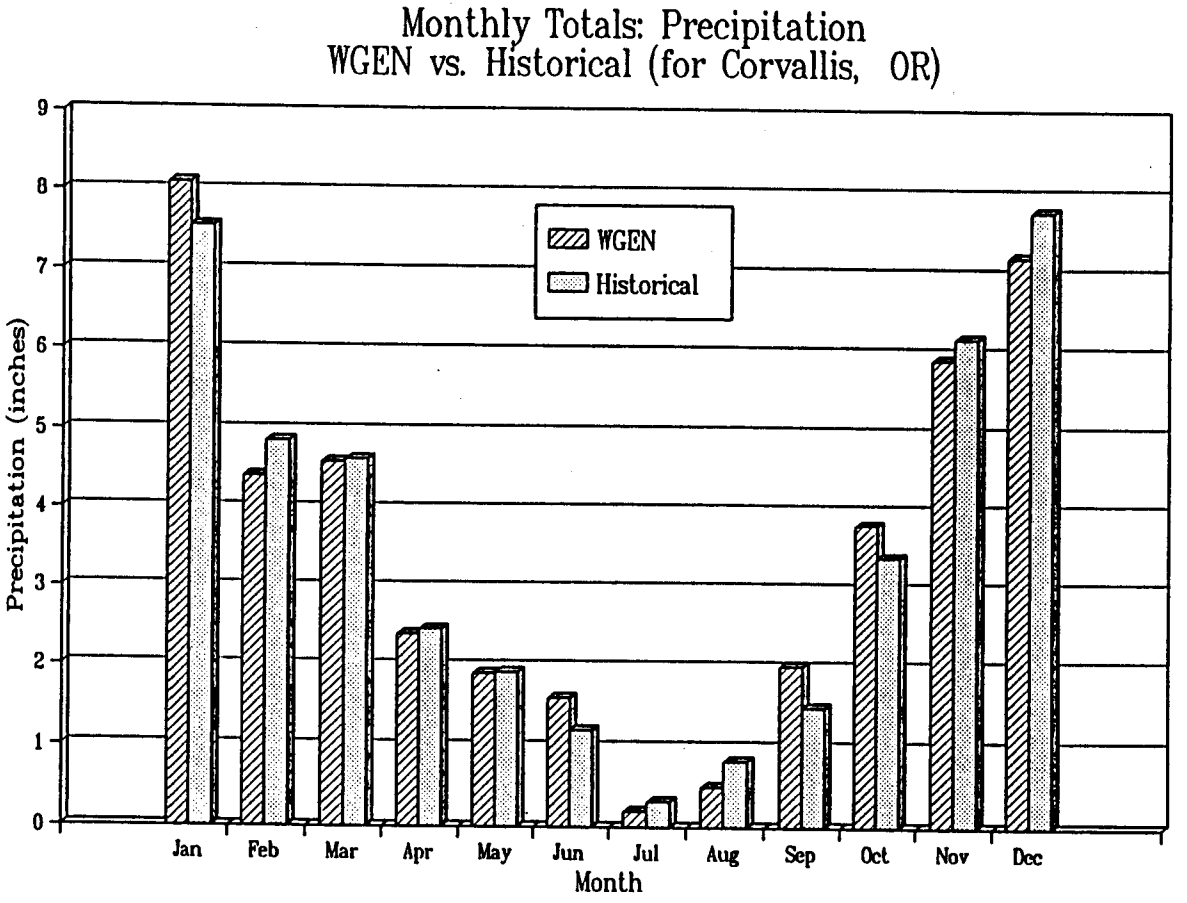


Figure 11. Monthly totals of precipitation for Corvallis, Oregon from WGEN and historical records.

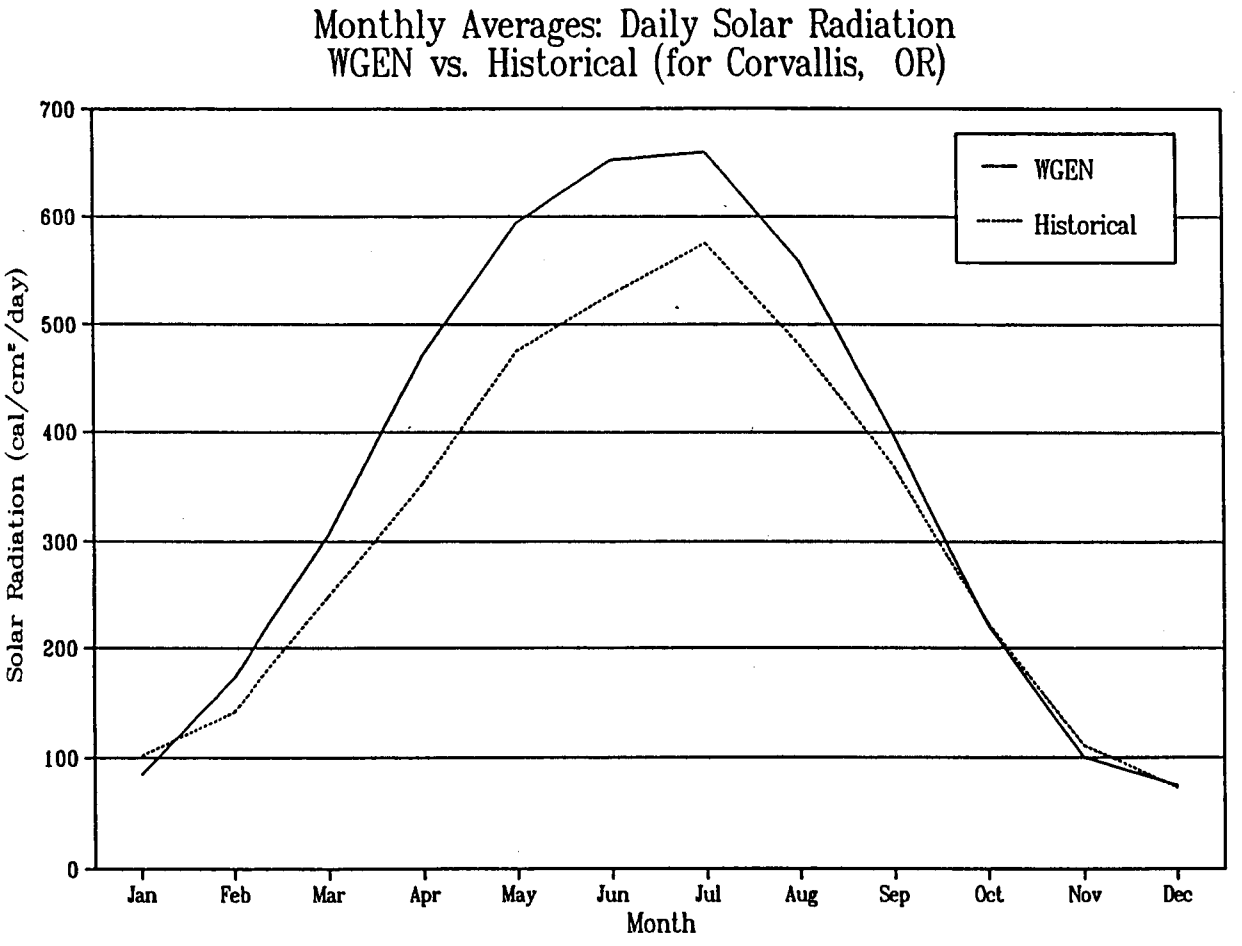


Figure 12. Monthly averages of daily total solar radiation for Corvallis, Oregon from WGEN and historical records.

Table 9. Monthly means of maximum temperature from WGEN and historical records, differences between the means, and P-value from t-test on the differences.

MAX. TEMPERATURE	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
WGEN	37.4	41.3	49.7	61.6	73.9	82.1	87.9	85.0	75.7	63.0	50.2	42.2
Historical	45.1	50.4	53.8	59.3	66.2	72.6	80.7	80.5	75.5	64.3	52.3	46.5
Difference	-7.7	-9.1	-4.1	2.3	7.7	9.5	7.2	4.5	0.2	-1.3	-2.1	-4.3
P-value	0.897											

Table 10. Monthly means of minimum temperature from WGEN and historical records, differences between the means, and P-value from t-test on the differences.

MIN. TEMPERATURE	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
WGEN	26.7	29.1	33.8	39.4	45.8	50.1	52.7	51.3	46.5	40.6	33.9	29.5
Historical	32.9	35.0	36.1	38.8	43.2	48.3	50.6	50.7	47.7	41.7	37.2	34.6
Difference	-6.2	-5.9	-2.3	0.6	2.6	1.8	2.1	0.6	-1.2	-1.1	-3.3	-5.1
P-value	0.1372											

Table 11. Monthly totals of precipitation from WGEN and historical records, differences between the totals, and P-value from t-test on the differences.

PRECIPITATION	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
WGEN	8.1	4.4	4.6	2.4	1.9	1.6	0.2	0.5	2.0	3.8	5.9	7.2
Historical	7.6	4.9	4.6	2.5	1.9	1.2	0.3	0.8	1.5	3.4	6.2	7.8
Difference	0.55	-0.46	-0.03	-0.06	-0.02	0.4	-0.11	-0.31	0.52	0.41	-0.27	-0.57
P-value	0.9416											

Table 12. Monthly means of daily total solar radiation from WGEN and historical records, differences between the means, and P-value from t-test on the differences.

SOLAR RADIATION	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
WGEN	85.6	172.2	305.6	471.0	594.4	651.7	658.8	560.1	397.6	221.8	99.3	74.5
Historical	102.0	141.3	248.5	353.9	475.0	527.7	575.9	481.1	369.8	224.3	111.5	72.4
Difference	-16.44	30.9	57.088	117.08	119.43	123.96	82.913	79.05	27.813	-2.525	-12.19	2.125
P-value	0.0069											

differences.

The input parameters obtained from the tables and charts in Richardson and Wright (1984) were checked to make sure they were properly entered. Then these parameters were altered until the generated data more accurately resembled the historical data. However, the altered parameters were very different from the parameters suggested by Richardson and Wright for the Corvallis area.

Because several examples of WGEN producing reasonable data for other specific sites are given in Richardson and Wright (1984), it was assumed that much of the problem described here was the tables and charts of input information did not adequately reflect the climate of the Willamette Valley in Oregon.

WGEN includes the ability to correct generated temperature values by using historical temperature data from the area for which data is being generated. When this option was used, the generated temperature means nearly exactly matched the historical data. Although this option is useful, it does not allow for correction of the solar radiation data, and, as mentioned before, a grower may not have access to historical data for his or her area.

Data Manager Program

The weather data manager program occupies about 98 kilobytes of disk space. Although it will fit on a floppy

disk, installing the program on a fixed disk allows it to run faster and provides more space for the files created by the program. Although microcomputers with large capacity fixed disks are common today, a more space-efficient method of storing these files would be beneficial.

There are eleven files created directly and indirectly by the manager program:

1) Program information file -- used to store information that is specific for a site and for a year of data collection: name, location, longitude, and latitude of weather station; year data collection started; and names of all other files used.

2) Communications file -- created by Campbell TELCOM program to store information used to communicate with the weather station's datalogger.

3) Raw data file -- created by the Campbell TELCOM program to store weather data retrieved from the station's datalogger; deleted after the data are stored in the back-up file.

4) Back-up data file -- Used to sequentially store all hourly, daily, and precipitation data sets retrieved from the datalogger.

5) QC extreme values file -- used to store the extreme values used in QC procedures.

6) Previous data file -- used to store the raw data from the previous day and the previous two hours for

comparison tests in QC.

7) File of current IBSNAT-formatted data -- used to store daily summaries of collected weather data in the IBSNAT format.

8) File of current and generated IBSNAT-formatted data -- same as above file but a complete year of weather data is created by appending the appropriate generated data to the existing collected data.

9) WGEN input file -- used to store parameters required to run WGEN for a specific location.

10) WGEN daily summary file -- used to store the generated daily summary data for specified number of years.

11) WGEN yearly summary file -- used to store the monthly means and totals for each year of data generated.

It was assumed that nearly all of the functions in this data manager program will be required for any weather station. Although some functions of this program were written to work with data specifically from the Campbell 012 station, these functions could be rewritten to work with data from other weather stations. Also, two programs from the Campbell PC-208 set of software are used by this program. Different programs would be required if a different brand of weather station were used.

Each of the five major sections of the data manager

program will be discussed in detail. Flow charts are provided for clarity.

1. Main Body (Figure 13)

The main body of the program lets the user choose what he or she wants to do and then calls the appropriate function. When this program is integrated into the DSS, the four main options on this menu will likely be options on a "pull-down" menu in the user-interface.

One additional function of the main body is to load the program information (file names, etc.) from their file when the program is first started. If the file of information does not exist, the user is immediately brought to the "edit program information" portion of the program (described later) and must enter the necessary data before continuing.

2. Retrieve Weather Data and Run QC (Figure 14)

These two functions are combined so that any time a set of weather data is retrieved from the station, it will automatically be run through the QC procedures. Data retrieval is achieved by using the TELCOM program contained in the Campbell PC-208 software ordered with the weather station. The TELCOM program is spawned with parameters directing it to immediately obtain any unretrieved sets of data from the weather station and store them in a comma-delimited ASCII file.

The appropriate files are then opened: the raw data

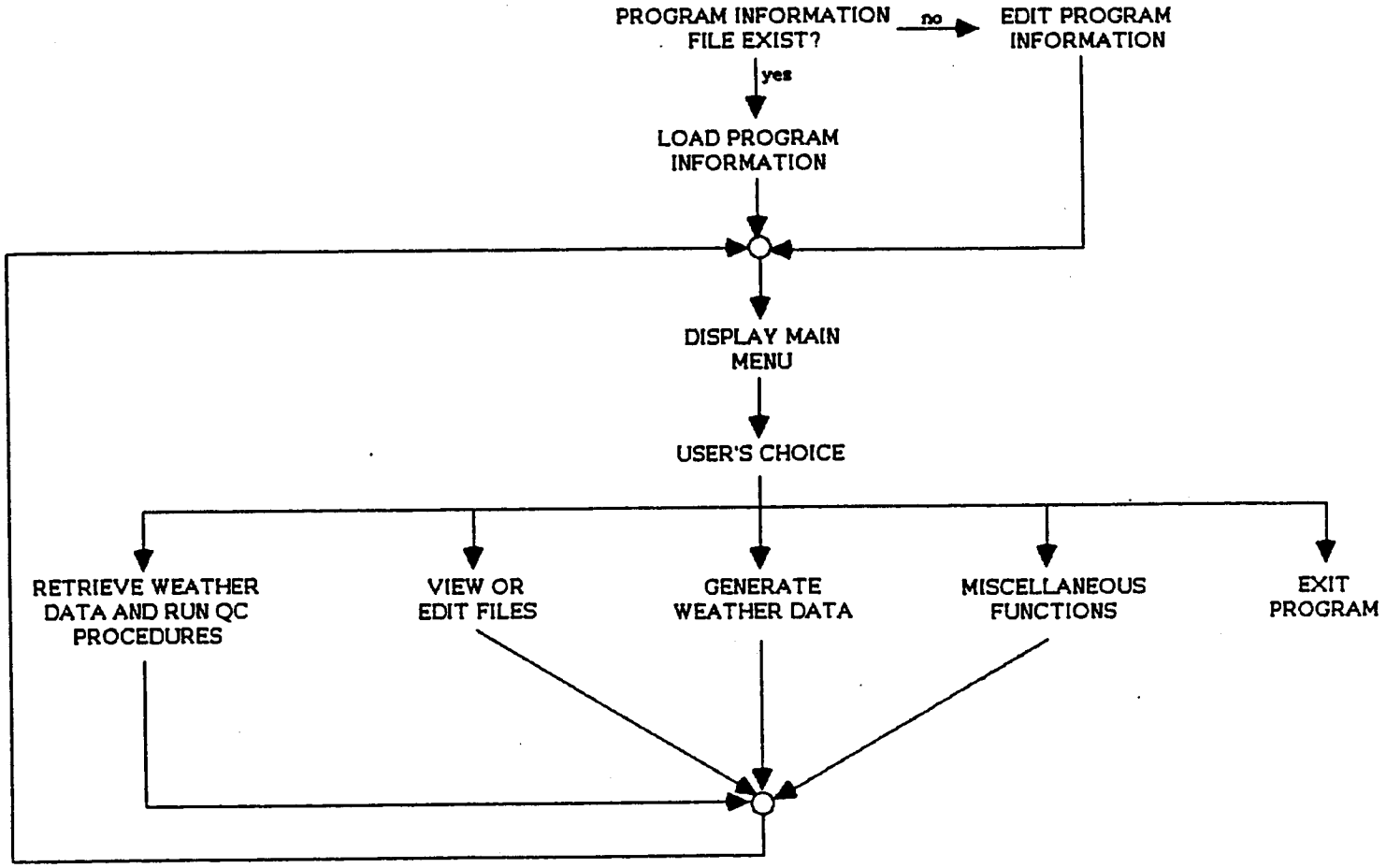


Figure 13. Flowchart for main body of program.

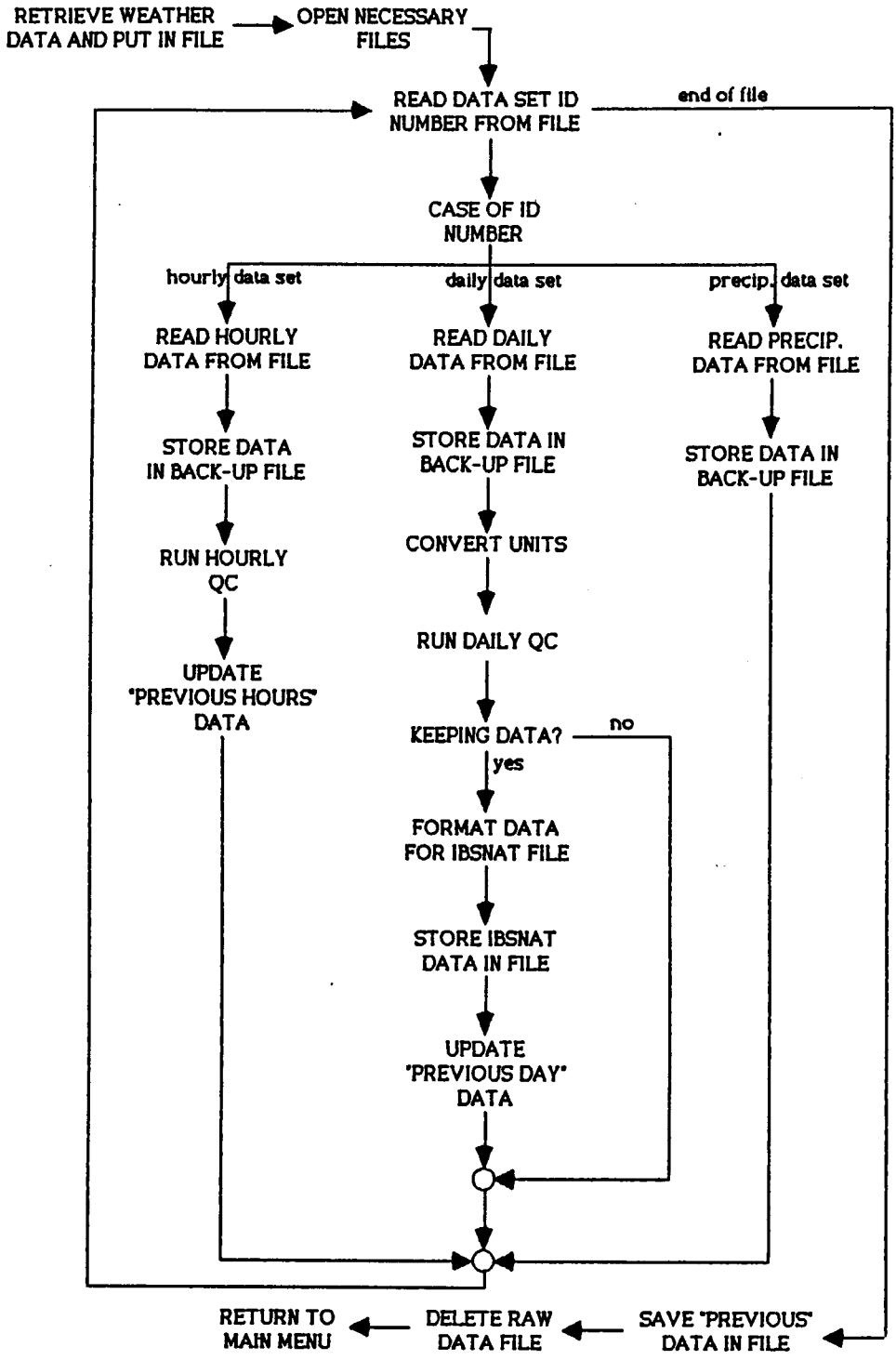


Figure 14. Flowchart for retrieval of weather data and performing QC procedures.

file, the data back-up file, the QC extremes file, the "previous" data file, and the file of current IBSNAT data. The raw data file is read one line at a time, with each line consisting of either an hourly, daily, or precipitation data set. The ID number of the data set is first read to determine its type and then program flow is directed accordingly.

For hourly data sets, the rest of the data set following the ID number is read from the file and stored in a struct variable. The data are then stored in the back-up file and run through the QC procedures (described in detail later). Finally, hourly data used by the comparison tests in QC are updated: the "two-hours previous" data set is deleted and replaced with the "one-hour previous" data set and the present hour's data set is then stored as the "one-hour previous" data set. The program then loops back to read the next data set in the raw data file.

The procedures for daily data sets are similar. The data are read from the raw file into a struct variable and then stored in the back-up file. Next, QC is performed on the data (detailed later). If the QC procedures detected a type of error which allows the user to choose whether or not to add the data set to the IBSNAT file, program flow will differ depending on the user's choice. If the "do not add" option is chosen, the subsequent procedures for daily data sets are skipped and the program loops back to read the next

data set in the raw file. Otherwise, the data to be placed in the IBSNAT file are converted to the proper units and then placed in the file. Finally, the "one-day previous" data set is deleted and replaced by this present day's data set. The program then returns to read the next data set in the raw file.

Conditional precipitation data sets are simply read from the raw file and stored in the back-up file. The program then loops back to read the next data set.

When the end of the raw data file is reached, the "previous" data sets used by QC are saved in a file and all other files are closed. The raw data file is then deleted (since the data is backed-up) and the program returns to the main menu.

QC of Hourly Data Sets (Figure 15). There is one set of tests for each of the five hourly values tested. Each set contains the one or more tests shown in Table 8. If the value being tested fails any one test in its set, the value fails QC. Otherwise, the value passes. For all sets of tests except that for Julian date, if a value fails QC, a warning is given to the user stating the date and time the value was recorded, the value itself, a description of the value, and why the value failed QC. In addition to the above warning, if a Julian date fails QC, the date is changed to day one to avoid later complications. Program flow continues when the user presses a key, signifying that

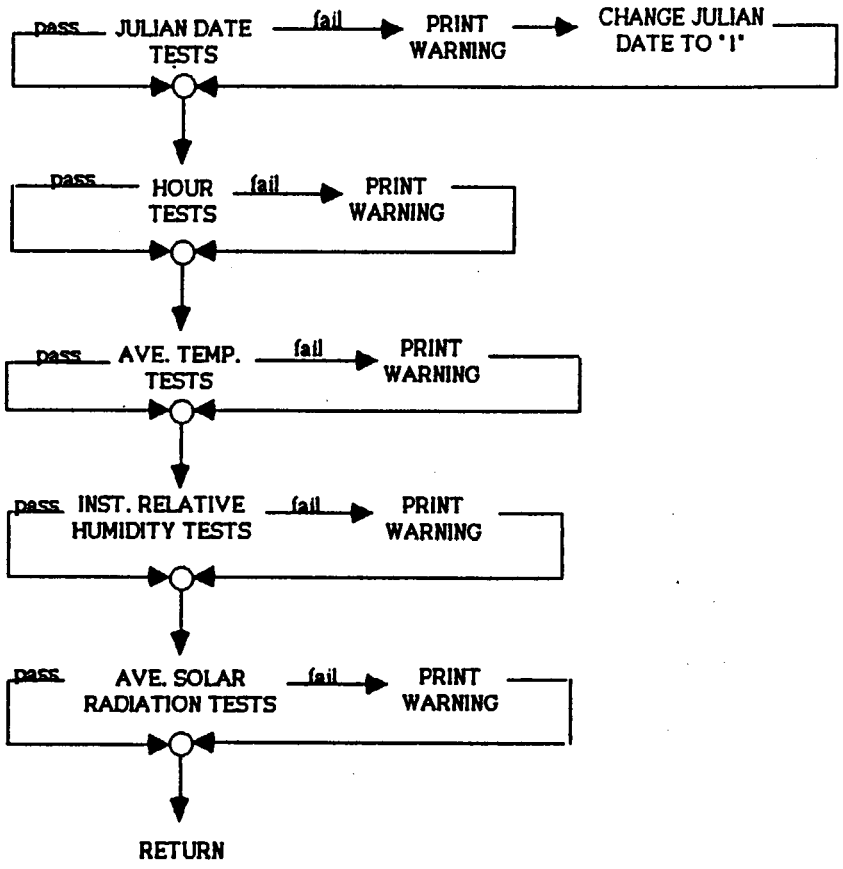


Figure 15. Flowchart for hourly QC procedures.

the warning has been read.

QC of Daily Data Sets (Figure 16). The ten daily values listed in Table 8 are tested. If a value fails any one of its tests, it fails QC. Failed values are handled differently depending on the value.

The QC procedures for the Julian date are complex, in order to ensure that the IBSNAT file used by the model will have no missing or duplicated days of data. For clarity, a flow chart of the Julian date QC procedures by themselves is shown in Figure 17.

The extreme value check is made first. If the date fails this test, the data set is shown to the user and the user may choose to delete that day's data (not store the data in the IBSNAT file) or change the date. (Changing the date causes the program to run through the extreme value check again with the new date.) These two options are provided because: 1) if the date is an errant number, the rest of the data will possibly be errant and should be deleted, or 2) if the data look reasonable, the user can just change the date to the proper day to keep the data.

Next, the difference between the date and the previous day's date is calculated. If this difference is equal to one day, then nothing else is done and the program continues with QC of the rest of the daily data set. If the difference is equal to zero or a negative number of days, the user may choose either to delete the day's data or

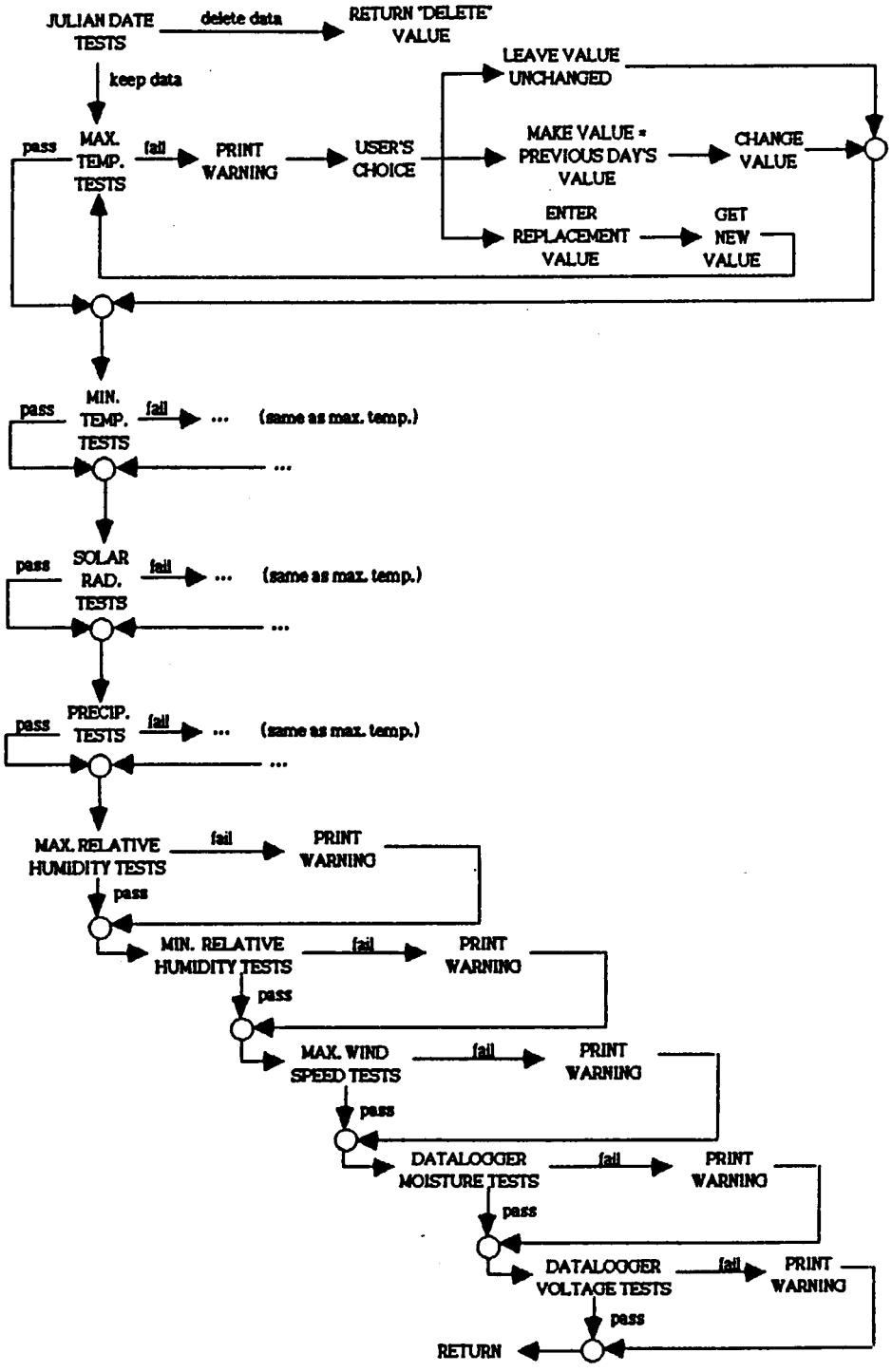


Figure 16. Flowchart for daily QC procedures.

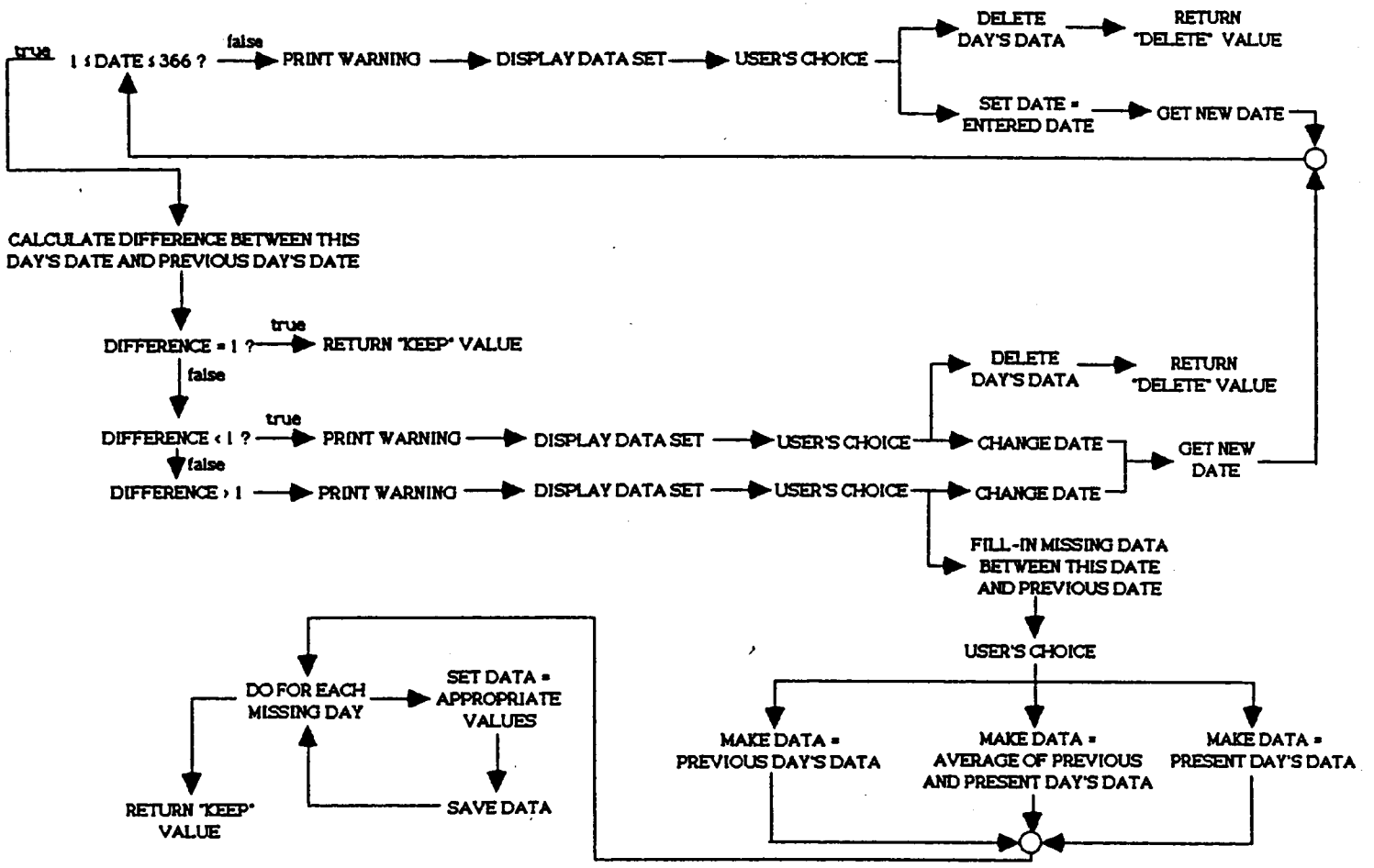


Figure 17. Flowchart for Julian date portion of daily QC procedures.

change the date, as described above. These two options are provided because: 1) power may have failed on the datalogger and the data for the day power was restored may not be reasonable and should be deleted, or 2) if the power failed and the data for the day power was restored are reasonable, the user would want to retain them, or 3) if the datalogger's date was improperly reset following a power loss, the data may be normal but the date must be changed to the proper day.

If the difference between the date being checked and the previous day's date is greater than one day, then data are missing for "difference-minus-one" days. The user may change the date or fill-in the missing day(s) of data. Changing the date causes the program to rerun QC on the new date. Data for the missing day(s) can be created quickly by having the user choose to make the missing data equal to the previous day's data, the present day's data, or an average of the two days. If more accurate data can be obtained for the missing day(s), they can be entered later through the editor portion of the program. These options of changing the date or adding missing data are provided for when: 1) the datalogger's date is improperly set following a power loss, or 2) data was not collected for a day or more because the weather station was not functioning, or 3) a day's data were deleted through one of the above listed options.

After checking the date, the four data values to be

used by the crop model are tested (maximum temperature, minimum temperature, precipitation, and solar radiation). The procedure for handling data which failed QC is exactly the same for all four of these values. A warning is given to the user stating the date and time the value was recorded, the value itself, a description of the value, and why the value failed QC. Also, for the purpose of comparison, the same value recorded the previous day is presented to the user. If the value failed an extreme value test, the high and low extremes used in the test are presented as an aide to the user for deciding if the value is reasonable. Finally, the user is asked to choose what to do with the value: 1) leave the value unchanged; 2) make the value equal to the value for the previous day; or 3) enter a replacement value. If a replacement value is entered, the QC tests are repeated using the new value.

For maximum temperature, minimum temperature, precipitation, solar radiation, and Julian date, any values that failed QC are "flagged" by placing a code on the same line as the data when it is placed in the IBSNAT file of current data. The first number of the code corresponds to the test that the value failed. The second number of the code corresponds to the action the user took on the failed value. A summary of the error and action codes is shown in Table 13. These flags provide an easy way for a person to detect the possible errant data in a file which mainly

Table 13. Descriptions of the error and action codes used to flag data placed in the current data file.

Julian date errors:

2 Missing one or more daily summaries prior to this date.
Actions: 1 Date changed
2 Data for missing day(s) filled in

3 The date for this data is not valid.
Actions: 1 Date changed
2 Data deleted

4 The date for this day's data is out of sequence.
Actions: 1 Date changed
2 Data deleted

5 Data for this date was missing.
Actions: 1 Data like last day of data in current file
2 Data like first day after missing data
3 Data averaged from these two days

All other QC errors have the same following actions:
1 Value left unchanged
2 Value made equal to previous day's value
3 New value entered by user

6 Max. temperature too high or too low.
7 Max. temperature constant over time.
8 Max. temperature change over one day is greater than allowable.
9 Max. temperature is less than min. temperature.
10 Max. temperature was edited by user.

11 Min. temperature too high or too low.
12 Min. temperature constant over time.
13 Min. temperature change over one day is greater than allowable.
14 Min. temperature was edited by user.

15 Total solar radiation too high or too low.
16 Total solar radiation was edited by user.

17 Total precipitation too high or too low.
18 Total precipitation was edited by user.

consists of good data.

3. View or Edit Files (Figure 18)

The functions on this menu provide a quick, line-by-line viewing of four major files and editing of the QC extremes file and the IBSNAT current data file.

Viewing Files. All four files are viewed in the same manner. After selecting the file to view, the user may choose to send the file to the screen or a printer. If the screen is chosen, successive lines of the selected file are printed on the screen until the screen is full. The user must then press a key to obtain the next screen of data. Data is printed in this manner until the end-of-file is reached, at which time the user must press a key before the program returns to the view/edit menu.

If the file is sent to the printer, each line of the file is printed until the end-of-file is reached. The program then returns to the view/edit menu.

Editing IBSNAT Current Data File. After choosing to edit this file, the user may either enter the date of the data to change or return to the view/edit menu. When a date is entered, it is first checked to make sure it is a valid date. If an invalid date is entered, the user is warned and must re-enter the date. Otherwise, the data for that date is located in the file and is displayed on the screen. Then the user may choose one of the displayed values to edit and

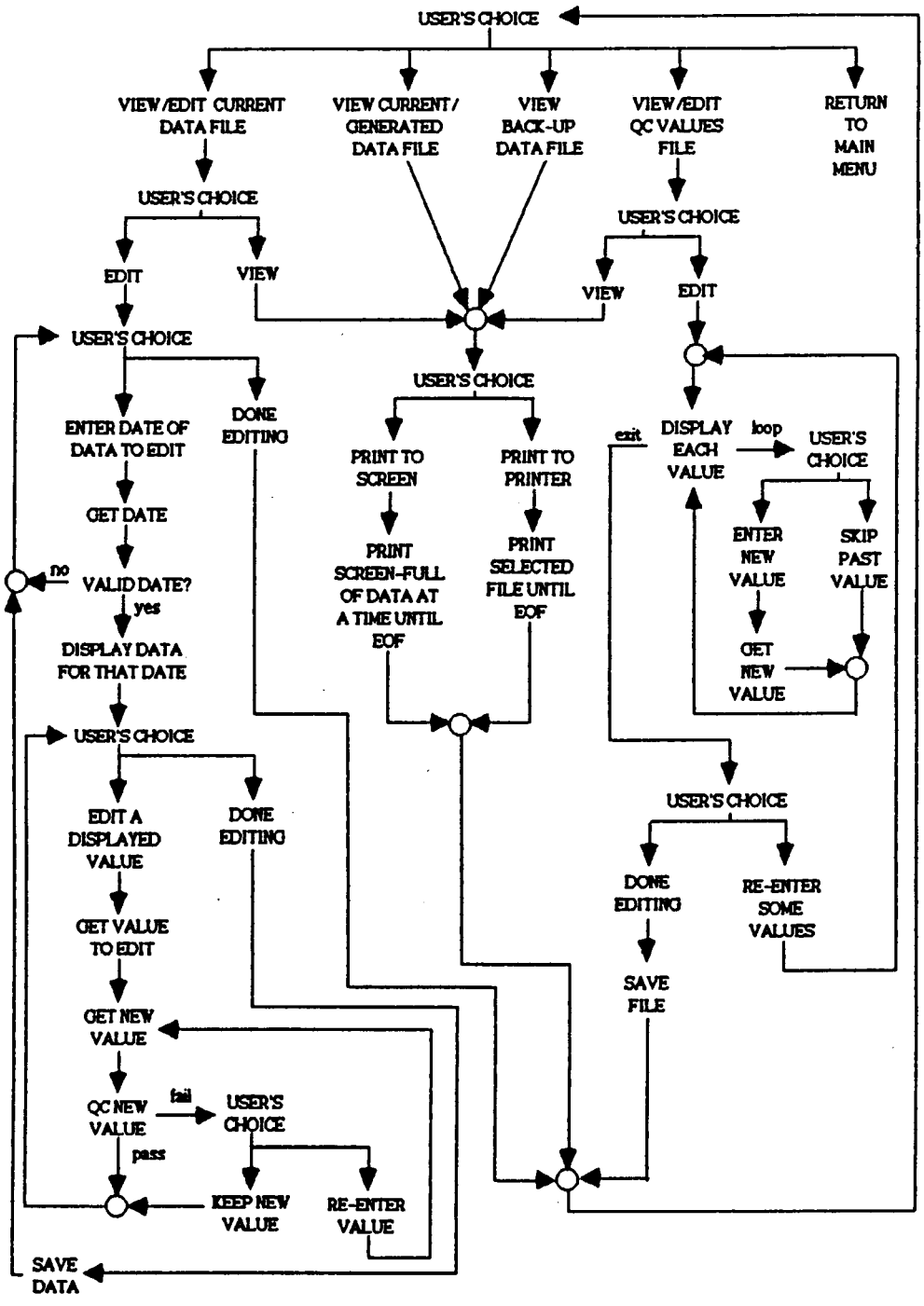


Figure 18. Flowchart for viewing and editing of files.

enter the new value. The new value is taken through the appropriate QC procedures. If the value fails QC, the user may choose to retain the value or re-enter it. Retaining the value has the same effect as if the value had passed QC, where the program returns to allow the user to choose another of the displayed values for editing or to end editing. If the user chooses to re-enter the value, the user enters the new value and it is run through QC. The QC tests are employed here mainly to help detect typing errors.

After a value has been edited, the error/action codes for the value are updated to reflect the edit. The action code is set to "replaced value." If an error code already existed for the value, it is left unchanged. If no error code existed, it is set to show that the value has been edited (see Table 13).

When the user signifies that all editing is finished for the given date, the data is saved and the user may enter another date for editing or quit editing and return to the view/edit menu.

Editing QC Extremes File. When the user chooses to edit the QC extremes file, the file is opened and the existing values are stored in variables. If the file does not exist, all of the values are initialized to zero. The user is then prompted to enter values one at a time. The value to be entered is identified and the existing value is given in brackets. The user may enter a new value or simply

press the "return" key to keep the existing value. Once the last value has been replaced or retained, the user may choose to repeat the edit to correct any mistakes or quit editing. When the user quits the editing, the extreme values are saved in the file and the program returns to the view/edit menu.

4. Generate Weather Data (Figure 19)

After choosing this option from the main menu, the user is given the option to edit the input parameters used to run WGEN. If the parameters are to be edited, the user can change any or all of the parameters in a manner similar to the editing of the QC values described previously. If the user chooses not to edit the parameters but the file of parameters does not exist, then the user must perform the edit before continuing.

The next series of statements determines if there are any data in the file of current weather data. If there is not at least one day's summary data in the file, the user may choose to exit this part of the program or to create an IBSNAT file which consists only of a year's worth of generated data. This latter option is provided for times when a user may wish to experiment with the crop model when a weather station is not connected to the system. To obtain this generated data file, a valid starting date for the data must be entered, then the current/generated data file (which

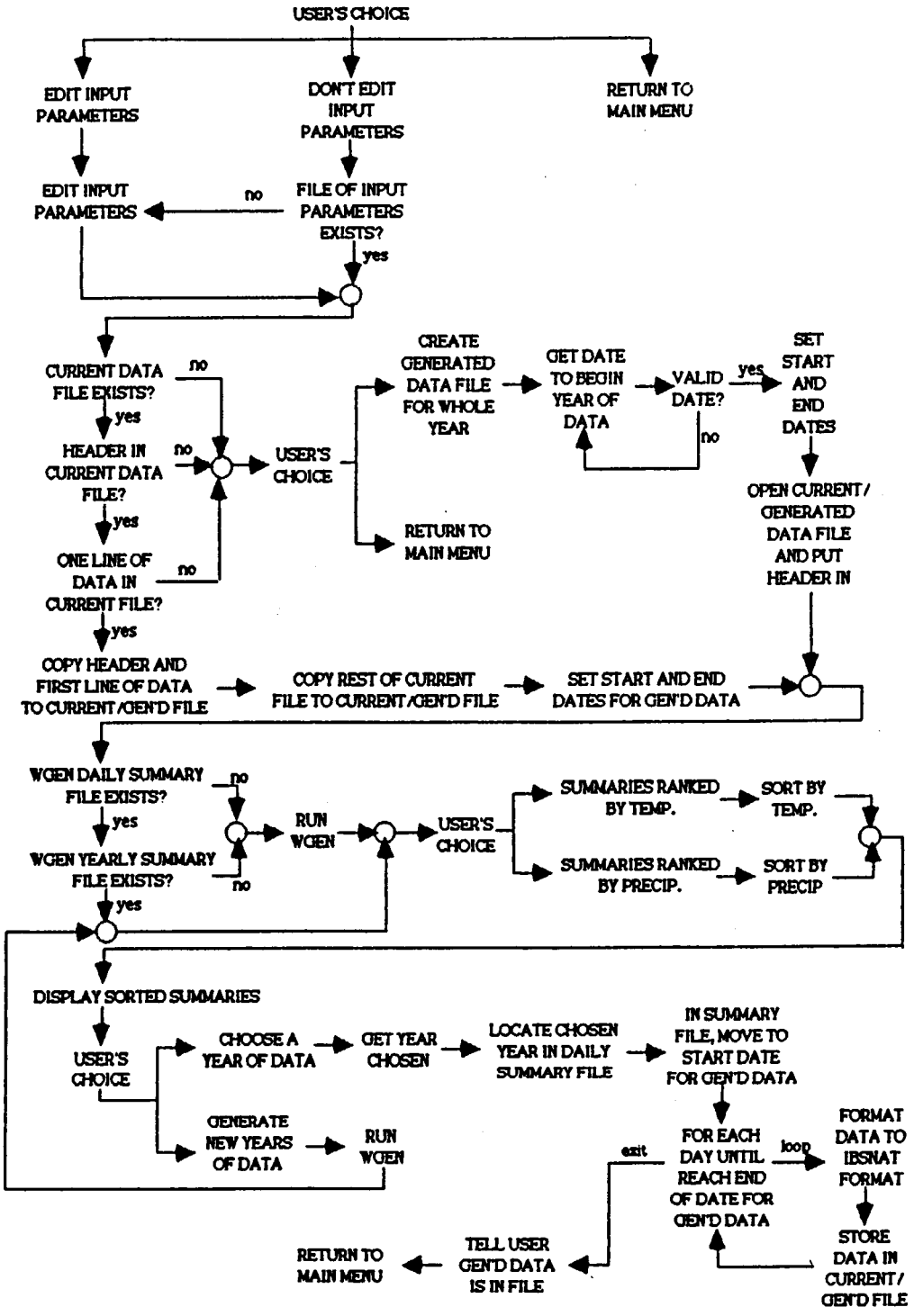


Figure 19. Flowchart for generating weather data.

is really only a generated data file in this case) is created and the line of identification data is copied to the file using the program information entered by the user (described in a later section).

If at least one daily summary exists in the current data file, the contents of the file are copied to the current/generated data file. The starting date for generated data is set as the day following the last day of current data. The ending date for generated data is set as one year after the first day of current data.

Program flow from this point is the same regardless of whether a complete or partial year of generated data is being put into the file.

Two checks are made to determine if the WGEN file of daily summary data exists and if the WGEN file of yearly summary data exists. If either file is missing, the WGEN program is run to create new summary files. Twelve years of generated data are produced when WGEN is run. Because each year of data is different, the user can choose data which best reflects the long-term weather pattern desired. Ideally, graphs of each generated year's monthly precipitation and temperature should be displayed to the user to allow quick comparisons between the different years. At the present time the method is used of displaying the total precipitation and the average temperature for each year, as described below.

For each year of data generated, total precipitation and average temperature are calculated for the period determined by the starting date and ending date for generated data. For example, if the first day of current data in the current file is for September 10, 1990 and the last day of current data is for January 25, 1991, then the first day for generated data is January 26, 1991 and the last day for generated data is September 10, 1991. So, for each of the twelve years of generated data, the total precipitation and average temperature are calculated for the period of January through September. These calculations are performed using the monthly data in the yearly summary file.

The user is then given the option to rank these twelve summaries either by precipitation or by temperature. After the data are sorted in descending order according to the chosen factor, they are displayed: the sorted factor displayed first and the corresponding non-sorted factor displayed adjacently in brackets. The user may then choose one of the years of data or choose to run WGEN again to produce twelve new years of data.

When a year of data is chosen by the user, the selected data are located in the daily summary file. Then each day's data to be put in the current/generated file are input from the daily summary file, converted to the IBSNAT format, and stored in the current/generated file. Finally, the user is informed that the required data have been placed in the

current/generated data file, and the program returns to the main menu.

5. Miscellaneous Functions (Figure 20)

Monitor Station. This function allows the user to monitor the weather station as it is collecting data and also to set the time and date in the weather station's datalogger. The former function is provided to help determine if the station is operating properly. The latter function is necessary when first getting the weather station running and following any power outages to the weather station. These functions are provided in the TERM program, which is part of the Campbell PC-208 software. The TERM program is spawned by the data manager program after the user is given a brief statement about using TERM. Most importantly, the user must know that setting the datalogger's clock requires that the clock in the computer on which the data manager program is running must be properly set. The user may return to the miscellaneous functions menu if he or she does not wish to enter TERM. Likewise, the user is returned to this menu after using the TERM program.

Edit Program Information. Information used by the data manager that the user may change are: name of weather station; name for current/generated data file used by the crop model; two-letter station ID, two-letter location ID,

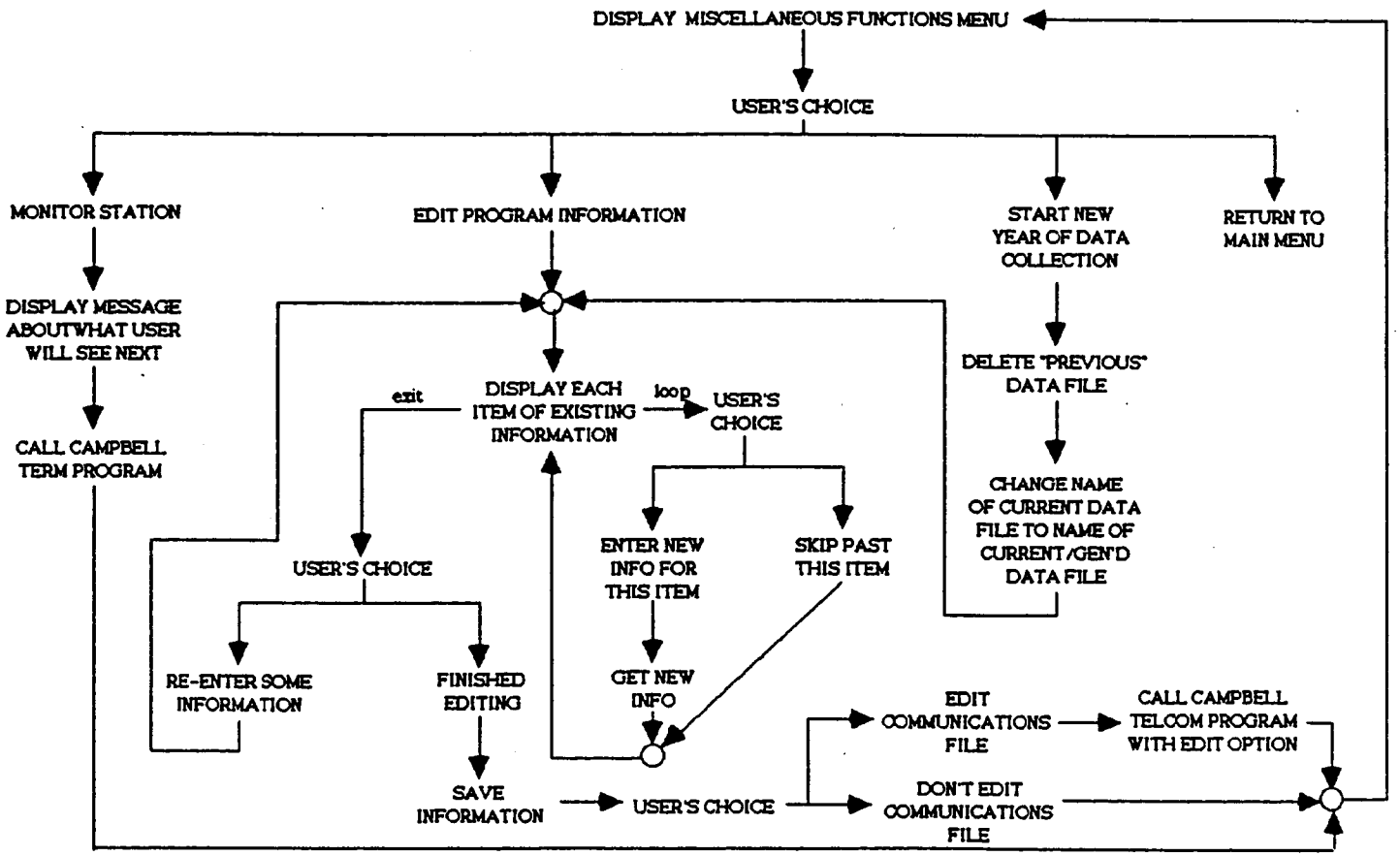


Figure 20. Flowchart for miscellaneous functions.

latitude, longitude, and the year data collection started (all required to make IBSNAT-format weather files). Names for the other files used by the program are created automatically by combining the weather station name and year of data collection and adding a different two- or three-letter extension for each different file.

When this option is chosen from the menu, the user may only edit the latitude and longitude. The other information may be edited only when starting the program for the first time (explained previously) and when starting a new year of data collection (explained later).

The information is edited in a manner similar to the editing of the QC values, described previously. When the user has finished editing and the information has been saved, the option is presented of editing the information file used by the Campbell software for communicating with the datalogger. This file must be created when first setting up the weather station and program, but should not need changed afterwards. However, the option is included if ever needed again. The file is edited through the Campbell TELCOM program and is a simple fill-in-the-blanks format. After editing of the file is completed (or after choosing not to edit the file) the user is returned to the miscellaneous functions menu.

Start New Year of Data Collection. This function causes weather data to stop being stored in the existing

files and initiates new weather data files. This procedure would normally be performed following harvest of the existing crop and prior to planting of the next year's crop.

After this option is selected, two functions are performed automatically: the file of "previous" data is deleted and the name of the file of current data is changed to the name of the current/generated data file. The user is then brought to the "edit program information" section. All the information the user should need to change is the name for the new current/generated data file. The year has been incremented automatically and all other information should remain unchanged. The editing procedure is exactly as described previously. However, one additional statement is executed which renames the old QC extreme values file to the new QC file name. This action prevents the user from having to re-enter the QC values. The user is returned to the miscellaneous functions menu when the editing is completed.

CONCLUSIONS

The development of a crop decision support system at OSU is currently underway. Several steps towards the completion of this system have been accomplished through this research.

An automated weather station was obtained and tested in the field. Although the station performed adequately, regular maintenance and sensor calibration were found to be essential.

A set of quality control procedures was developed for data from the automated station. These procedures are performed automatically and require little input from the user.

A weather generator program, WGEN, was chosen to provide possible future weather data from the present day to the end of the current season. The program produces the required data for use by the crop models without the need of historical weather data. A test of the program showed that the program input parameters suggested by the WGEN authors were not valid for Corvallis, Oregon.

Finally, a weather data manager program was written. This menu-driven program retrieves data from the automated station, performs the QC procedures on the data, allows the user to edit the data, and formats and stores the data in a file usable by the IBSNAT crop models. The program also

runs the weather generator and formats the resulting data for use by the models.

Some additions to the program that might be beneficial are: graphs for viewing the collected weather data and for choosing generated data; automatic unit conversions so a grower may choose either English or SI units and then would only have to view and enter data in those units; and ability to change the date and year of each daily summary when editing the current data file, in case an improper date was somehow given to a summary.

Much work remains before the DSS is usable. The expert system is being developed. It must be able to provide worth-while management strategies for a grower to consider. The user-interface must yet be written to make the DSS a friendly and informative program. The CERES-Wheat model is currently being modified. It must be able to run for one day and then stop and store all necessary data, and it should also be able to be easily modified as more is learned about modeling the crop. Programs that simulate pest damage should also be sought out or developed to enhance the DSS.

I would like to see a prototype of the DSS tested, as soon as is feasible, at Hyslop Farm or at the East Farm (which is closer to campus) or at both sites (because of differing soil types). I also think it would be beneficial to have growers experiment with the prototype, as they can provide valuable information from a user's standpoint.

BIBLIOGRAPHY

Ashcroft, G. L., G. D. McCurdy, G. E. Bingham, and E. Malek. 1990a. Quality control of automated-climate-station data. Paper presented at the 1990 International Summer Meeting sponsored by the American Society of Agricultural Engineers. Columbus, OH, 24-27 June, 1990. ASAE, St. Joseph, MI.

Ashcroft, G. L., G. D. McCurdy, S. M. Oborn, and G. E. Bingham. 1990b. Integrating diverse climatic data into a unified database. Paper presented at the 1990 International Summer Meeting sponsored by the American Society of Agricultural Engineers. Columbus, OH, 24-27 June, 1990. ASAE, St. Joseph, MI.

Campbell Scientific, Inc. 1989. 012 weather station user's manual. Campbell Scientific, Inc., Logan, UT.

Day, W. 1984. Wheat growth and modelling: an introduction. p. 1-5. *In* W. Day and R. K. Atkin (ed.) Wheat Growth and Modelling. Plenum Press, NY.

Ferguson, J. 1990. Mid-season cotton update. Ag Consultant. 46(8):18-19.

Godwin, D. C. and P. L. G. Vlek. 1985. Simulation of nitrogen dynamics in wheat cropping systems. p. 311-332. *In* W. Day and R. K. Atkin (ed.) Wheat Growth and Modelling. Plenum Press, NY.

The International Benchmark Sites Network for Agrotechnology Transfer (IBSNAT). 1986. Documentation for IBSNAT crop model input and output files, version 1.0. Tech. Rep. 5. IBSNAT, University of Hawaii, Manoa, HA.

Jones, J. W., R. F. Colwick, and E. D. Threadgill. 1972. A simulated environmental model of temperature, evaporation, rainfall, and soil moisture. Trans. of the American Society of Agricultural Engineers. 15(2):366-372.

Jones, J. W., P. Jones, and P. A. Everett. 1987. Combining expert systems and agricultural models: a case study. Trans. of the American Society of Agricultural Engineers. 30:1308-1314.

Jones, J. W., S. S. Jagtap, G. Hoogenboom, and G. Y. Tsuji. 1989. The structure and function of DSSAT. p. 1-14. *In*

IBSNAT Symposium: Decision Support System for Agrotechnology Transfer, Part 1, Symposium Proceedings, 81st annual meeting of the ASA, Las Vegas, NV, 17 October, 1989. IBSNAT, University of Hawaii, Manoa, HA.

Legg, B. J. 1981. Aerial environment and crop growth. p. 129-149. *In* D. A. Rose and D. A. Charles-Edwards (ed.) Mathematics and Plant Physiology. Academic Press, London.

Lemmon, H. 1986. COMAX: An expert system for cotton crop management. Science. 233:29-33.

Nicks, A. D. and J. F. Harp. 1980. Stochastic generation of temperature and solar radiation data. Journal of Hydrology. 48:1-17.

Palmer, R. G. 1986. How expert Systems can improve crop production. Agricultural Engineering. 67:28-29.

Plant, R. E. 1989a. An integrated expert decision support system for agricultural management. Agricultural Systems. 29(1):49-66.

Plant, R. E. 1989b. An artificial intelligence based method for scheduling crop management actions. Agricultural Systems. 31(1):127-155.

Plant, R. E., F. G. Zalom, J. A. Young, and R. E. Rice. 1989. CALEX/Peaches, and expert system for the diagnosis of peach and nectarine disorders. HortScience. 24(4):700.

Reinke, B. C. and S. E. Hollinger. 1990. Quality control of near real-time meteorological data. Paper presented at the 1990 International Summer Meeting sponsored by the American Society of Agricultural Engineers. Columbus, OH, 24-27 June, 1990. ASAE, St. Joseph, MI.

Richardson, C. W. and D. A. Wright. 1984. WGEN: A model for generating daily weather variables. ARS-8. USDA-ARS.

Ritchie, J. T. 1985. A user-oriented model of the soil water balance in wheat. p. 293-305. *In* W. Day and R. K. Atkin (ed.) Wheat Growth and Modelling. Plenum Press, NY.

Ritchie, J. T., D. C. Godwin, and S. Otter-Nacke. 1985. CERES-Wheat: A simulation model of wheat growth and development. Unpublished draft.

Samani, Z. A. and Hargreaves, G. H. 1987. Crop production models in water management. p. 66-74. *In* L. G. James and M. J. English (ed.) Irrigation systems for the 21st Century. American Society of Civil Engineers, NY.

Tang, A., J. W. Jones, D. Imamura, and G. Y. Tsuji. 1989. Decision support system for agrotechnology transfer. Poster #104. *In* IBSNAT Symposium: Decision Support System for Agrotechnology Transfer, Part 2, Poster Session, 81st annual meeting of the ASA, Las Vegas, NV, 17 October, 1989. IBSNAT, University of Hawaii, Manoa, HA.

Uehara, G. 1985. The International Benchmark Sites Network for Agrotechnology Transfer (IBSNAT). p. 271-274. *In* W. Day and R. K. Atkin (ed.) Wheat Growth and Modelling. Plenum Press, NY.

Whisler, F. D., B. Acock, D. N. Baker, R. E. Fye, H. F. Hodges, J. R. Lambert, H. E. Lemmon, J. M. McKinion, and V. R. Reddy. 1986. Crop simulation models in agronomic systems. Advances in Agronomy. 40:141-208.

Zuzel, J. and R. Karow. 1988. WEATHERWIZARD user's guide. Spec. Rep. 831. Oregon State University Extension Service, Corvallis, OR.

APPENDICES

USER'S GUIDE FOR

THE WEATHER DATA MANAGER PROGRAM

written by

William S. Donaldson

WEATHER DATA MANAGER PROGRAM

This program is designed to be part of the future Oregon State University agricultural decision support system. The main functions of the program are retrieval, quality control, and formatting of raw data from a field weather station to a file which may be used by IBSNAT crop models.

The data manager program may also be used to place generated weather data in an IBSNAT-format file. This data may be used to create a file consisting of a year's worth of generated data only. Additionally, the generated data may be placed in the file of collected weather data, following the last day of collected data, thereby creating a complete year of weather data.

The other functions of the program are viewing and editing of selected data files and monitoring of the weather station while it is in operation.

SYSTEM REQUIREMENTS

- IBM PC or compatible computer with a fixed-disk drive.
- Campbell Scientific, Inc. 012 weather station linked via modem to the above computer.
- Campbell Scientific, Inc. PC-208 software.

CONTENTS OF PROGRAM DISK

- W.EXE weather data manager program.
- WGENPC.EXE weather generator program.

FILES CREATED BY THE PROGRAM

There are eleven files created directly and indirectly by the manager program:

1) Program information file -- used to store information that is specific for a site and for a year of data collection: name, location, longitude, and latitude of weather station; year data collection started; and names of all other files used.

2) Communications file -- created by Campbell TELCOM program to store information used to communicate with the weather station's datalogger.

3) Raw data file -- created by the Campbell TELCOM program to store weather data retrieved from the station's datalogger; deleted after the data are stored in the back-up file.

4) Back-up data file -- Used to sequentially store all hourly, daily, and precipitation datasets retrieved from the datalogger.

5) QC extreme values file -- used to store the extreme values used in QC procedures.

6) Previous data file -- used to store the raw data from the previous day and the previous two hours for

comparison tests in QC.

7) File of current IBSNAT-formatted data -- used to store daily summaries of collected weather data in the IBSNAT format.

8) File of current and generated IBSNAT-formatted data -- same as above file but a complete year of weather data is created by appending the appropriate generated data to the existing collected data.

9) WGEN input file -- used to store information to run WGEN for a specific location.

10) WGEN daily summary file -- used to store the generated daily summary data for specified number of years.

11) WGEN yearly summary file -- used to store the monthly means and totals for each year of data generated.

NECESSARY INFORMATION TO HAVE AVAILABLE

- Monthly extremes (maximum and minimum values) for your area for temperature (degrees F), daily total solar radiation (MJ/m2), and precipitation (inches). If these values are not available, reasonable estimates will suffice. Data conversion factors are given in Appendix A.
- Maximum and minimum values for relative humidity (usually 0 and 100) and for datalogger moisture and

battery voltage (given in the CSI 012 weather station operator's manual).

- Maximum amount (in degrees F) of change in temperature from one day to the next.
- Latitude and longitude.
- Name of weather station (up to six characters).
- Name for file of current/generated weather data (later becomes a year's worth of collected weather data).
- Weather station ID and location ID (both two characters).
- Values for running the weather generator from the tables and charts given in Appendix B; values nearest your location should be used.

SET-UP AND START-UP

Copy the files on the program disk to the desired directory on your fixed disk. Install in the same directory at least the following programs from the CSI PC-208 software: TERM, TELCOM.

From this same directory, type "w <enter>" to begin the program.

OVERVIEW

This section is meant to provide a quick description of how the data manager program is used. Phrases in bold print are portions of the program which are explained in detail following this overview.

When the program is entered for the first time, the file of program information must first be created. You are brought immediately to the **Edit Program Information** portion of the program to create this file and a datalogger communications file. After these functions are completed, you are brought to the main menu.

The datalogger's clock must be set at this point, if it has not already been set. Move to the Miscellaneous Functions menu and choose the **Monitor Station/Set Datalogger Clock** option to accomplish this.

Next, move to the View/Edit Files menu and choose the edit portion of the **View/Edit File of QC Values** option to create the file of QC values.

The last information which must be entered is the information used by the weather generator. Choose the **Run Weather Generator** option and answer "y" when asked if you want to edit the input file. After entering the necessary information, you may return to the main menu or you may run the weather generator and create an IBSNAT file consisting only of a year's worth of generated data (if no daily summaries have been collected from the weather station yet).

After the weather station has collected an hour or more of data, you may retrieve the data by choosing the **Retrieve Data, Run QC, Add Data to IBSNAT File** option.

You may view the collected data by moving to the View/Edit Files menu and choosing the **View File of Weather Station Back-up Data** option. If daily summaries were collected, you may view them by choosing the view portion of the **View/Edit File of Current Data** option. You may edit the values in a daily summary by choosing the edit portion of this option and entering the date of the summary to edit.

Generated data may be added to the file of collected (or current) data by choosing the **Run Weather Generator** option from the main menu. The resulting file of current and generated data may be viewed by choosing the **View File of Current/Generated Data** option from the View/Edit Files menu.

You may monitor the output of the weather station by choosing the **Monitor Station/Set Datalogger Clock** option under the Miscellaneous Functions menu. Also under this menu, is the option to choose when a season of weather data collection is finished and a new one is to begin. This **Start New Year of Data Collection** option will allow you to change information for the new collection year.

DETAILED FUNCTION DESCRIPTIONS

Retrieve Data, Run QC, Add Data To IBSNAT File

Purpose: Retrieve data from weather station which has accumulated since the last collection time; run quality control procedures on this data; and add daily summaries of this data in IBSNAT format to the current weather data file.

This option is chosen from the main menu. You are asked if you really want to perform this function. Answer 'n' if you do not (you will be returned to the main menu) or answer 'y' to continue.

The weather station is called using the CSI TELCOM program and any uncollected data are retrieved (some numbers will appear on the screen as data is retrieved). If communication with the weather station was not possible, a message will appear saying "Datalogger does not respond!" and you will likely get another message saying "Could not open weather station raw data file!" and will be returned to the main menu. If a communication problem occurs, check the datalogger communications file, the modems, and all cable connections to discover possible problems.

If data was retrieved without a problem, a raw data file was created and the program now begins to read in each data set collected, store them in the back-up file, perform the necessary QC procedures on them, and format and store the IBSNAT data in the current data file. If the message

"Insufficient data in QC file!" appears at this time, you will be returned to the main menu and must create a QC values file (**View/Edit File of QC Values**).

Quality control procedures are provided to help you detect any problems which might occur with the weather station and to ensure that the IBSNAT file created using this weather data is complete and reasonable for your area. Remember that these procedures only help identify problems after they have occurred (such as sensors freezing or not operating because of various activities of animals). Regular checking (about once per week or per every other week) of the weather station's sensors is essential to producing a quality set of weather data.

QC of the data varies depending on the type of data set. Precipitation data sets have no QC performed on them.

Hourly data sets have some QC performed. If an hourly value fails QC you will be notified by a message, similar to that shown below:

```
        --- Report from:  QC procedures on raw data from weather station ---

WARNING:  Relative humidity constant over time
   Recorded date:  Dec 8, 90, 600 hours
   Errant value:        96.40 (%)  Hourly Rel. Humidity
              (press any key)
```

A warning is given regarding why the value failed QC. The date and time the value was recorded is shown, along with the value itself, its units, and the type of value that it is. Because hourly data is not used in the IBSNAT file,

nothing more is done with the failed value. The warning is simply given to aid you in detecting any problems with the weather station. Pressing any key causes the program to continue.

Daily data sets have the most detailed QC procedures, because their data is used in the IBSNAT file. When the first daily summary is read by the program, you will see a notice similar to the one shown below:

```
        --- Report from:  QC procedures on raw data from weather station ---

NOTICE:  This is the first daily summary collected
   Recorded date:  Dec 7, 90
        Summary of this day's data:
         39.52 (degrees F)  Daily Max. Temp.
         35.89 (degrees F)  Daily Min. Temp.
          2.07 (MJ/m2/day)  Daily Total Solar Rad.
          0.00 (in)  Daily Total Precip.

Choose action to take:
        1.  Keep data as is
        2.  Delete this data
        3.  Change the date for this data
        (Must choose 2. or 3. if date is invalid)
Your choice:
```

The date of the data set is shown along with the data which will be placed in the IBSNAT file. You are presented three options to determine the fate of this data set. The options are provided to take into account the following situations: the data looks fine and the date is correct, so you would want to keep this data; or the weather station was started up in the afternoon and the daily summary does not take into account the morning's weather, so you would not want to keep this data; or the data looks fine, but the datalogger's clock was not set properly until after this summary was

collected, so you just need to change the date to the proper day. (If the datalogger had a problem, such that it gave a date to the data set which is not one of the 365 or 366 days of the year, "invalid date" would be shown for the recorded date; you must delete the data or change the date if this occurs.) If you do choose to change the date, the new date is run through the QC procedures and you will be presented with same notice again. Choose to keep the data as is if you are satisfied with the date you entered. The quality control of the rest of the data is continued after you choose to either "keep" or "delete" this first day's data.

There are three other types of messages given by the QC procedures for the daily data sets. The first is when one of the values to be placed in the IBSNAT file (max. and min. temperature, solar radiation, and precipitation) fails QC. A warning is given similar to the one shown below:

```
        --- Report from:  QC procedures on raw data from weather station ---

WARNING:  Max. temperature too high or too low
   Recorded date:  Dec 8, 90, 0 hours
   Errant value:        74.49 (degrees F)  Daily Max. Temp.
   Extreme values:      66.00      -14.00
   Yesterday's value:   39.52

Choose action to take:
   1.  Keep value the same
   2.  Change value to yesterday's value
   3.  Enter a replacement value
Your choice:
```

The warning format is like the warning given for failed hourly values, except for: if the value failed because it is out of the range given by the extreme values, the extreme values used are shown; and the value for the previous day is

also shown. You may keep the value as it is, change it to equal the value for the previous day, or enter a different value to replace it. If you enter a value, the QC procedures will be performed on that value. If the value fails QC, you may keep it or re-enter the value.

When daily values which are not used in the IBSNAT file fail QC, the warning you get is like the warning for a failed hourly value. Pressing any key causes the program to continue.

The final warning type is for problems with the date of the daily data set. A warning of this type is shown below:

```
        --- Report from:  QC procedures on raw data from weather station ---

WARNING:  Missing one or more daily summaries prior to this date
   Recorded date:  Dec 16, 90
        Summary of this day's data:
         44.06 (degrees F)  Daily Max. Temp.
         30.67 (degrees F)  Daily Min. Temp.
          2.51 (MJ/m2/day)  Daily Total Solar Rad.
          5.01 (in)  Daily Total Precip.

Last daily summary in current data file is for Dec 11, 90
        Summary of this day's data:
         67.45 (degrees F)  Daily Max. Temp.
         33.32 (degrees F)  Daily Min. Temp.
          3.72 (MJ/m2/day)  Daily Total Solar Rad.
          0.06 (in)  Daily Total Precip.

Choose action to take:
        1.  Change the date of the present day's dataset
        2.  Fill in data between these two dates
Your choice:
```

This type of warning will be printed when the date of the daily summary is not one day past the date of the previous day's summary (which is the last daily summary in the current data file). If the date is before or equal to the date of the last summary in the current data file, the date is said to be "out of order" and you must decide to either

change the date or delete the summary. If the date is more than one day past the date of the last summary, as in the above example, you may choose to change the date or to fill in data between the two dates. The latter option is provided in case the weather station was not operating for a period of time and data is missing for that period. If you do choose to fill in missing data, after the rest of the data in the daily summary is run through QC, you will be presented with the options of: 1)making the missing summaries equal to the last summary in the current data file; 2) making the missing summaries equal to the summary that was just run through QC (the "present day's data"); or 3) making the missing summaries equal to an average of the data for the last day in the current file and the present day's data. The missing summaries will be produced and stored in the current data file after you choose the desired option. If you wish to enter actual data (obtained from another nearby weather station) for the missing day(s), you may do so in the edit portion of the **View/Edit File of Current Data** portion of this program.

One final note on the QC procedures. Dates of daily summaries and values used in the IBSNAT file which fail QC are identified in the current data file. At the end of the line of data which contains (or contained) the failed value, two numbers separated by a comma are given. The first number corresponds to the value which failed QC and why it

failed. The second number corresponds to the action you (the user) took after being notified of the failed value. An example is shown below:

```
osst 90 353  5.27  -0.3  -9.0   0.0   0.00  Dec 19
osst 90 354  7.26  -7.8 -12.1   0.0   0.00  Dec 20  15,1
```

The data for December 19 did not fail any QC errors. However, for December 20, the solar radiation value was "too high or too low" and the user responded by leaving the value unchanged. Note that there can be up to five of these error/action sets after a daily summary (one for each of the date, max. temperature, min. temperature, precipitation, and solar radiation). These numbers provide a way to determine which values in the file may be questionable and if they have been changed. The possible errors and their associated actions are listed in Appendix C.

After the last data set collected from the weather station has been processed, you are returned to the main menu.

View/Edit File of Current Data

Purpose: To view the contents of the current data file and to edit the data values for any day in the current data file.

After choosing this option from the View/Edit Files menu, you are asked if you want to view the file or edit the

file. If viewing the file, you are next asked if you want to send the file to the screen or a printer. If the screen is chosen, successive lines of the selected file are printed on the screen until the screen is full. You may then press 'q' to quit viewing the file or press any other key to obtain the next screen of data. Data is printed in this manner until the end-of-file is reached, at which time you must press a key before the program returns to the View/Edit Files menu.

If the file is sent to the printer, each line of the file is printed until the end-of-file is reached. The program then returns to the View/Edit Files menu.

If you choose to edit this file, you may either enter the date of the data to change or return to the View/Edit Files menu. If you enter a date, it is first checked to make sure it is a valid date (you must re-enter an invalid date). If the date does not exist in the file, you will be told so, and then may enter another date.

When the data for that date is located in the file, it is displayed on the screen, as shown below:

```
        Data for Dec 19, 90
        CHOOSE A VALUE TO CHANGE

1.   -0.30 (degrees C)  Daily Max. Temp.
2.   -9.00 (degrees C)  Daily Min. Temp.
3.    5.27 (MJ/m2/day)  Daily Total Solar Rad.
4.    0.00 (cm)  Daily Total Precip.

Q.  Quit and Save Changes

Your Choice:
```

You may then choose one of the displayed values to edit and enter the new value. The new value is taken through the appropriate QC procedures. If the value fails QC, you may choose to retain the value or re-enter it. Retaining the value has the same effect as if the value had passed QC, where the program returns to allow you to choose another of the displayed values for editing or to end editing. If you choose to re-enter the value, the QC procedures are run again on the new value. The QC tests are employed here mainly to help detect typing errors which you may not detect.

Any values which you choose to change will be given an error/action set of codes similar to those described under **Retrieve Data, Run QC, Add Data to IBSNAT File.** The "error" code signifies that the value was edited and the action code signifies that you entered a new value. If the value you changed already had a set of error/action codes, the error code will remain as it was, but the action code will be set to signify that you entered a new value.

When you choose to quit and save changes, the data are saved and you may enter another date for editing or quit editing and return to the View/Edit Files menu.

View File of Current and Generated Data

Purpose: To view the contents of the current/generated data file.

Choose this option from the View/Edit Files menu. After choosing to send the file to the screen or printer, the file is viewed exactly as described previously for viewing the current data file under **View/Edit File of Current Data.**

View/Edit File of QC Values

Purpose: To view the contents of the QC values file and to edit these values.

Choose this option from the View/Edit Files menu. This file is viewed exactly as described previously for viewing the current data file under **View/Edit File of Current Data.** The first line of each set of values has a label describing which values they are. For the extreme values, the maximum is listed first, followed by the minimum. Monthly extremes are listed from January to December.

For editing this file, if the file does not exist, all of the values are initialized to zero. Otherwise, the existing values are obtained from the file. You are then prompted to enter values one at a time. The value to be entered is identified and the existing value is given in brackets. You may enter a new value or simply press the <enter> key to keep the existing value. Once the last value has been replaced or retained, you may choose to repeat the edit to correct any mistakes or quit editing. When you quit the editing, the values are saved in the file and the

program returns to the View/Edit Files menu.

View File of Weather Station Back-up Data

Purpose: To view the contents of the back-up data file.

Choose this option from the View/Edit Files menu. After choosing to send the file to the screen or printer, the file is viewed exactly as described previously for viewing the current data file under **View/Edit File of Current Data.**

Run Weather Generator

Purpose: To provide reasonable data for a site either to create an IBSNAT file of generated data only (to run a crop model for research or educational purposes) or to add the data to an existing file of actual weather data (to provide possible weather data for the rest of the season).

Which one of the above stated purposes you fulfill is determined by whether or not there is any weather data in the current data file. If there is no data in the file, you may create a file of only generated data. If there is data in the file, you may add generated data to the existing current data.

After choosing this option from the main menu, you are given the option to edit the input information used to run WGEN. If the information is to be edited, you may change

any or all of the information by entering new values after the prompts or pressing only the <enter> key (to keep the existing value). Obtain the necessary values for your area from the tables and charts given in Appendix B. If you choose not to edit the information but the file of information does not exist, then you must perform the edit and create the input information file before continuing.

Next, the program determines if there is at least one daily summary in the current data file. If not, you may choose to return to the main menu or create an IBSNAT file which consists of a year's worth of generated data only. If you choose to create the file, you must enter a valid starting date for the data. The ending date for the data is one year past this date. Even though this file will contain only generated data, when the current/generated data file is referred to, this is the file which will contain the generated data.

If at least one daily summary exists in the current data file, the contents of the file are copied to the current/generated data file. The starting date for generated data is set as the day following the last day of current data. The ending date for generated data is set as one year past the first day of current data.

Two checks are made to determine if the WGEN file of daily summary data exists and if the WGEN file of yearly summary data exists. If either file does not exist, the

WGEN program is run to create new summary files. Twelve years of generated data are produced when WGEN is run. Depending on the type of computer you are using, this may take anywhere from a few seconds to a few minutes. After WGEN is run once, it need not be run again unless you wish to have a different set of twelve years of data produced.

For each year of data generated, total precipitation and average temperature are calculated for the period from the generated data starting date to the ending date, as figured earlier. These calculations are performed using the monthly data in the WGEN yearly summary file.

You are then given the option to rank these twelve summaries either by precipitation or by temperature. After the data are sorted in descending order according to the chosen factor, they are displayed: the sorted factor displayed first and the corresponding non-sorted factor displayed adjacently in brackets. An example is shown below:

```
Summaries for the remainder of the season, ranked by
total precip. (ave. temperature in brackets)

      1.    34.8 inches     [ 53.1 F]
      2.    35.3 inches     [ 50.2 F]
      3.    35.5 inches     [ 51.8 F]
      4.    37.6 inches     [ 50.9 F]
      5.    37.7 inches     [ 50.2 F]
      6.    39.3 inches     [ 51.5 F]
      7.    40.4 inches     [ 50.6 F]
      8.    40.4 inches     [ 50.7 F]
      9.    41.6 inches     [ 51.7 F]
     10.    43.2 inches     [ 50.7 F]
     11.    47.3 inches     [ 52.1 F]
     12.    60.8 inches     [ 51.7 F]
     13.   Generate new data

Your choice:
```

You may then choose one of the years of data or choose to run WGEN again to produce twelve new years of data.

When a year of data is chosen, the selected data are located in the daily summary file. Then each day's data to be put in the current/generated file are input from the daily summary file, converted to the IBSNAT format, and stored in the current/generated file. When this is completed, you are informed that the required data have been placed in the current/generated data file, and the program returns to the main menu. The current/generated data file is then ready to be used by an IBSNAT crop model or you can view the file by choosing the **View File of Current and Generated Data** option from the View/Edit Files menu.

Monitor Station/Set Datalogger Clock

Purpose: To monitor the output of the weather station while it is in operation and to set the time and date of the weather station's datalogger using CSI's TERM program.

Choose this option under the Miscellaneous Functions menu. You are first given a brief message about using TERM. Most importantly, you must know that setting the datalogger's clock requires that the clock in your computer must be properly set. If you do not wish to enter TERM or need to set your computer's clock, you may choose to return to the Miscellaneous Functions menu. (You must exit the data manager program and use the DOS TIME and DATE functions

to set your computer's clock.)

If you choose to enter the TERM program, you may monitor the weather station by choosing option 'M' and may set the datalogger clock by choosing option 'K'. Choose 'Q' to quit TERM and return to the Miscellaneous Functions Menu.

Edit Program Information

Purpose: To allow user to enter the information used by the program when it is first started and, afterwards, to allow for editing of the latitude and longitude; also allows for creation and editing of datalogger communications file.

When the data manager is first started, the program information file must be created before anything else may be done. That file is actually created through this function, even though the program brings you directly here with out letting you choose the option.

The station name is used to identify many files used by the program and is used by the TELCOM and TERM programs to identify the weather station. The current/generated file is the one used by the IBSNAT models and, after a year of data has been collected, it will be the name given to the file containing the year of collected data. The station and location ID's, latitude, and longitude are used to format the current and current/generated files according to IBSNAT specifications. Also, the latitude is used as part of the input file for WGEN.

The program information is initialized to certain values. If you wish to change any of this information, you may do so as you are prompted or you may keep the existing data by pressing <enter>. You may continue to re-enter any or all of the values until you are satisfied with them and choose to not re-enter any values. You may not change anything except the latitude and longitude once you have left this portion of the program, unless you choose the **Start New Year of Data Collection** option.

After you have finished entering the program information, the values are saved, and you are brought to the TELCOM program to edit the datalogger communications file. It is important to set the proper datalogger type and the proper interface devices and baud rate. The data file format should always be set to "Comma Delineated ASCII" and the next time to call should always be set to some time far in the future so when TELCOM is exited you will not waste time by having the program call the station. A sample of what the options may look like is given below:

```
                    Station File Name: sta

          Datalogger or Command Type: CR10              Security Code: 0
              Data Collection Method: Since Last Call; Append File; 1st Area
Nbr of Arrays to Backup on First Call: 0
                  Data File Format: Comma Delineated ASCII
  Fix Datalogger Clock Using PC Clock: No
      Primary Call Interval (minutes): 1440
  Recovery Call Interval #1 (minutes): 15
  Repetitions of Recovery Interval #1: 4
  Recovery Call Interval #2 (minutes): 1440
Maximum Time Call Will Take (minutes): 10
                  Next Time To Call:  8/19/99 19:47:00

     Interface Devices:
     COM1                 Baud Rate: 1200
     Short Haul




     ^O = Save/Resume      ^P = Save/Done      ESC = Abandon Edit
```

Use the <enter> key to move through the options. The space bar will sometimes let you move through choices on individual options. Press ctrl-P to save the information when you are done editing. You are then returned to the Main menu.

The Edit Program Information option should not need to be used after the initial data entry. Only the latitude and longitude can be edited through this option, because changing the other data will cause several problems with file names, etc. However, you can use this option if you need to make corrections/changes to the datalogger communications file. Use the **Start New Year of Data Collection** option to change all of the program information.

After choosing this option, you are given the opportunity to return to the Miscellaneous Functions menu.

If you choose to continue, you may enter new values for the latitude and longitude or may press <enter> to keep the values as they are. When you are finished editing the values, they are saved and you are given the opportunity to edit the datalogger communications file. If you do not edit this file, you are returned to the Miscellaneous Functions menu, and if you do edit the file, you are returned to the Miscellaneous Functions menu after you are finished editing.

Start New Year of Data Collection

Purpose: To end a year of data collection and start a new year of data collection with different file names.

This option means weather data would stop being stored in the existing files and would begin being stored in a new set of files. This process would be performed following harvest of the existing crop and prior to planting of the next year's crop.

This option is chosen from the Miscellaneous Functions menu. You are given the choice to return to the Miscellaneous Functions menu or continue. When you continue, two functions are performed automatically: the file of "previous" data is deleted and the name of the file of current data is changed to the name of the current/generated data file. You are then brought to the **Edit Program Information** section. All the information you should need to change is the name for the new

current/generated data file. The year is incremented automatically and all other information should remain constant. One additional function renames the old QC values file to the new QC file name. This action prevents you from having to re-enter the QC values. You are returned to the Miscellaneous Functions menu when the editing is completed.

USER'S GUIDE APPENDIX A

DATA CONVERSION FACTORS

To convert °F to °C:

$$5/9 * (°F - 32) = °C$$

To convert °C to °F:

$$9/5 * °C + 32 = °F$$

To convert inches to centimeters:

$$inches * 2.54 = cm$$

To convert centimeters to inches:

$$cm/2.54 = inches$$

To convert cal/cm²/day (langleys) to MJ/m²:

$$cal/cm^2/day * 0.0419 = MJ/m^2$$

To convert MJ/m² to cal/cm²/day (langleys):

$$MJ/m^2/0.0419 = cal/cm^2/day$$

USER'S GUIDE APPENDIX B

WGEN INPUT INFORMATION

Tables and Charts reprinted from :

Richardson and Wright. 1984. WGEN: A model for generating daily weather variables. ARS-8. USDA-ARS.

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
BIRMINGHAM,AL	P(W/W)	0.491	0.505	0.475	0.444	0.530	0.481	0.548	0.426	0.480	0.395	0.457	0.495
	P(W/D)	0.264	0.299	0.285	0.245	0.183	0.220	0.307	0.265	0.175	0.144	0.213	0.267
	ALPHA	0.643	0.640	0.648	0.712	0.675	0.626	0.802	0.660	0.676	0.630	0.715	0.647
	BETA	0.710	0.765	0.845	0.724	0.662	0.699	0.499	0.629	0.744	0.716	0.593	0.769
MOBILE,AL	P(W/W)	0.419	0.483	0.514	0.340	0.419	0.547	0.593	0.515	0.538	0.444	0.375	0.493
	P(W/D)	0.294	0.286	0.257	0.197	0.202	0.280	0.446	0.351	0.232	0.135	0.193	0.271
	ALPHA	0.577	0.629	0.556	0.512	0.644	0.623	0.713	0.686	0.548	0.645	0.613	0.624
	BETA	0.766	0.816	0.969	1.434	0.902	0.799	0.697	0.774	1.109	0.659	0.628	0.894
MONTGOMERY,AL	P(W/W)	0.447	0.456	0.435	0.380	0.475	0.457	0.436	0.408	0.514	0.444	0.348	0.471
	P(W/D)	0.269	0.289	0.262	0.219	0.185	0.220	0.317	0.264	0.166	0.117	0.175	0.279
	ALPHA	0.713	0.691	0.699	0.634	0.634	0.706	0.620	0.762	0.546	0.601	0.684	0.681
	BETA	0.525	0.680	0.786	0.852	0.681	0.589	0.648	0.408	1.179	0.767	0.619	0.687
FLAGSTAFF,AZ	P(W/W)	0.558	0.470	0.483	0.464	0.362	0.490	0.545	0.515	0.438	0.470	0.495	0.536
	P(W/D)	0.114	0.138	0.151	0.127	0.073	0.051	0.254	0.279	0.132	0.082	0.114	0.115
	ALPHA	0.895	0.889	0.854	0.945	0.983	0.592	0.826	0.782	0.659	0.811	0.689	0.729
	BETA	0.327	0.292	0.318	0.257	0.187	0.423	0.283	0.324	0.452	0.347	0.436	0.510
PHOENIX,AZ	P(W/W)	0.407	0.478	0.364	0.303	0.294	0.313	0.366	0.318	0.429	0.354	0.327	0.400
	P(W/D)	0.085	0.077	0.070	0.042	0.018	0.022	0.099	0.147	0.057	0.054	0.060	0.078
	ALPHA	0.825	0.822	0.998	0.883	0.899	0.629	0.752	0.650	0.532	0.680	0.917	0.746
	BETA	0.225	0.182	0.242	0.199	0.140	0.271	0.233	0.335	0.462	0.310	0.220	0.323
YUMA,AZ	P(W/W)	0.273	0.077	0.250	0.176	0.000	0.000	0.238	0.211	0.313	0.318	0.222	0.349
	P(W/D)	0.056	0.048	0.041	0.024	0.008	0.000	0.030	0.052	0.017	0.025	0.038	0.047
	ALPHA	0.841	0.763	0.998	0.517	0.802	0.000	0.637	0.670	0.394	0.686	0.624	0.882
	BETA	0.180	0.205	0.102	0.332	0.127	0.000	0.248	0.253	0.875	0.327	0.276	0.197
FORT SMITH,AR	P(W/W)	0.426	0.444	0.394	0.479	0.445	0.407	0.421	0.341	0.432	0.366	0.423	0.444
	P(W/D)	0.157	0.216	0.238	0.280	0.245	0.210	0.195	0.171	0.171	0.134	0.147	0.165
	ALPHA	0.655	0.701	0.719	0.709	0.658	0.632	0.580	0.650	0.752	0.625	0.638	0.719
	BETA	0.447	0.501	0.574	0.624	0.796	0.674	0.762	0.730	0.604	0.956	0.803	0.534
LITTLE ROCK,AR	P(W/W)	0.489	0.437	0.500	0.498	0.500	0.480	0.401	0.383	0.396	0.367	0.392	0.462
	P(W/D)	0.217	0.267	0.242	0.270	0.190	0.179	0.233	0.177	0.174	0.154	0.186	0.225
	ALPHA	0.619	0.681	0.790	0.686	0.554	0.651	0.703	0.581	0.624	0.659	0.633	0.665
	BETA	0.699	0.708	0.564	0.730	1.090	0.664	0.600	0.710	0.909	0.628	0.823	0.694
BAKERSFIELD,CA	P(W/W)	0.425	0.482	0.346	0.474	0.287	0.444	0.300	0.250	0.214	0.391	0.364	0.303
	P(W/D)	0.132	0.132	0.130	0.095	0.039	0.008	0.010	0.006	0.019	0.022	0.082	0.117
	ALPHA	0.966	0.827	0.845	0.822	0.841	0.805	0.800	0.796	0.893	0.967	0.999	0.913
	BETA	0.175	0.215	0.162	0.214	0.115	0.112	0.090	0.063	0.135	0.255	0.232	0.155

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
BLUE CANYON,CA	P(W/W)	0.731	0.678	0.663	0.631	0.556	0.488	0.067	0.296	0.370	0.437	0.628	0.710
	P(W/D)	0.208	0.213	0.231	0.184	0.155	0.073	0.025	0.032	0.054	0.090	0.200	0.174
	ALPHA	0.716	0.808	0.880	0.721	0.798	0.742	0.996	0.439	0.600	0.567	0.710	0.791
	BETA	1.597	1.053	0.798	0.777	0.463	0.350	0.070	0.615	0.456	1.694	1.188	1.432
EUREKA,CA	P(W/W)	0.754	0.693	0.724	0.615	0.518	0.398	0.122	0.306	0.397	0.529	0.691	0.718
	P(W/D)	0.331	0.265	0.261	0.209	0.167	0.128	0.064	0.058	0.095	0.177	0.272	0.266
	ALPHA	0.837	0.758	0.968	0.777	0.743	0.777	0.998	0.499	0.651	0.851	0.718	0.877
	BETA	0.556	0.506	0.331	0.359	0.295	0.150	0.050	0.289	0.275	0.379	0.622	0.510
FRESNO,CA	P(W/W)	0.509	0.519	0.393	0.477	0.340	0.158	0.050	0.050	0.154	0.286	0.484	0.475
	P(W/D)	0.172	0.156	0.140	0.105	0.056	0.024	0.010	0.010	0.017	0.034	0.098	0.154
	ALPHA	0.724	0.759	0.852	0.752	0.998	0.998	0.998	0.698	0.848	0.862	0.827	0.755
	BETA	0.384	0.333	0.313	0.357	0.134	0.076	0.080	0.095	0.187	0.287	0.351	0.336
MT SHASTA,CA	P(W/W)	0.718	0.675	0.646	0.591	0.563	0.466	0.258	0.378	0.386	0.490	0.628	0.689
	P(W/D)	0.233	0.211	0.206	0.154	0.137	0.101	0.042	0.049	0.049	0.097	0.200	0.185
	ALPHA	0.776	0.650	0.729	0.706	0.834	0.998	0.998	0.944	0.558	0.635	0.660	0.623
	BETA	0.724	0.782	0.461	0.471	0.284	0.182	0.150	0.188	0.607	0.593	0.842	0.962
SAN DIEGO,CA	P(W/W)	0.580	0.388	0.427	0.465	0.396	0.190	0.050	0.333	0.368	0.250	0.479	0.458
	P(W/D)	0.124	0.131	0.139	0.106	0.047	0.026	0.006	0.010	0.019	0.046	0.103	0.111
	ALPHA	0.683	0.659	0.737	0.734	0.867	0.998	0.998	0.617	0.847	0.578	0.785	0.708
	BETA	0.398	0.392	0.301	0.235	0.084	0.064	0.040	0.233	0.223	0.230	0.318	0.373
SAN FRANCISCO,CA	P(W/W)	0.662	0.602	0.566	0.515	0.429	0.250	0.091	0.238	0.280	0.385	0.587	0.680
	P(W/D)	0.225	0.193	0.203	0.121	0.063	0.042	0.016	0.030	0.028	0.090	0.168	0.166
	ALPHA	0.725	0.762	0.762	0.803	0.744	0.512	0.900	0.769	0.486	0.535	0.702	0.761
	BETA	0.550	0.385	0.338	0.329	0.199	0.254	0.150	0.083	0.420	0.478	0.423	0.487
COLORADO SPRINGS,CO	P(W/W)	0.333	0.400	0.467	0.456	0.530	0.487	0.521	0.559	0.423	0.424	0.366	0.329
	P(W/D)	0.098	0.123	0.173	0.159	0.232	0.235	0.400	0.253	0.140	0.111	0.098	0.087
	ALPHA	0.905	0.998	0.850	0.656	0.601	0.607	0.708	0.755	0.716	0.774	0.885	0.988
	BETA	0.077	0.068	0.114	0.264	0.361	0.380	0.300	0.278	0.302	0.224	0.141	0.070
DENVER,CO	P(W/W)	0.423	0.384	0.503	0.483	0.540	0.443	0.435	0.373	0.419	0.408	0.427	0.394
	P(W/D)	0.130	0.177	0.201	0.202	0.208	0.246	0.237	0.228	0.149	0.113	0.122	0.126
	ALPHA	0.781	0.853	0.790	0.655	0.611	0.637	0.634	0.600	0.693	0.690	0.948	0.988
	BETA	0.118	0.152	0.179	0.292	0.453	0.295	0.333	0.278	0.292	0.312	3.149	0.093
GRAND JUNCTION,CO	P(W/W)	0.407	0.410	0.388	0.404	0.476	0.427	0.318	0.384	0.391	0.475	0.385	0.344
	P(W/D)	0.173	0.183	0.179	0.168	0.107	0.086	0.114	0.184	0.136	0.107	0.127	0.169
	ALPHA	0.947	0.994	0.998	0.849	0.821	0.835	0.764	0.794	0.840	0.983	0.918	0.973
	BETA	0.096	0.089	0.093	0.128	0.150	0.155	0.121	0.189	0.176	0.172	0.131	0.099

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
PUEBLO,CO	P(W/W)	0.362	0.411	0.455	0.404	0.455	0.417	0.370	0.417	0.301	0.372	0.292	0.435
	P(W/D)	0.104	0.113	0.136	0.116	0.172	0.180	0.246	0.230	0.143	0.092	0.093	0.071
	ALPHA	0.935	0.998	0.966	0.634	0.650	0.693	0.720	0.615	0.661	0.719	0.839	0.928
	BETA	0.066	0.065	0.100	0.327	0.322	0.227	0.294	0.346	0.246	0.322	0.141	0.091
WINDSOR LKS,CT	P(W/W)	0.406	0.454	0.445	0.475	0.412	0.469	0.356	0.387	0.444	0.421	0.513	0.493
	P(W/D)	0.311	0.311	0.301	0.310	0.309	0.295	0.275	0.274	0.236	0.182	0.287	0.297
	ALPHA	0.780	0.650	0.755	0.688	0.725	0.667	0.702	0.594	0.556	0.641	0.687	0.694
	BETA	3.555	0.485	0.487	0.504	0.369	0.464	0.467	0.718	0.750	0.694	0.530	0.506
WILMINGTON,DE	P(W/W)	0.450	0.410	0.451	0.482	0.462	0.393	0.401	0.420	0.437	0.428	0.460	0.476
	P(W/D)	0.263	0.282	0.312	0.318	0.291	0.244	0.251	0.244	0.172	0.162	0.245	0.226
	ALPHA	0.783	0.727	0.732	0.771	0.692	0.674	0.578	0.684	0.592	0.667	0.699	0.746
	BETA	0.335	0.435	0.468	0.377	0.364	0.537	0.774	0.655	0.852	0.550	0.514	0.494
DISTRICT OF COLUMBIA	P(W/W)	0.424	0.415	0.452	0.478	0.455	0.377	0.400	0.441	0.406	0.394	0.361	0.410
	P(W/D)	0.265	0.254	0.303	0.276	0.260	0.269	0.243	0.231	0.178	0.162	0.242	0.244
	ALPHA	0.834	0.811	0.828	0.789	0.751	0.622	0.581	0.607	0.635	0.628	0.731	0.679
	BETA	0.299	0.384	0.367	0.383	0.423	0.604	0.793	0.810	0.645	0.610	0.478	0.508
JACKSONVILLE,FL	P(W/W)	0.401	0.398	0.408	0.320	0.477	0.564	0.555	0.584	0.598	0.505	0.330	0.370
	P(W/D)	0.212	0.253	0.190	0.172	0.181	0.294	0.391	0.342	0.320	0.200	0.157	0.191
	ALPHA	0.677	0.731	0.626	0.670	0.586	0.651	0.676	0.613	0.622	0.545	0.665	0.677
	BETA	0.486	0.670	0.693	0.676	0.770	0.800	0.706	0.926	0.795	0.869	0.419	0.500
MIAMI,FL	P(W/W)	0.328	0.364	0.286	0.345	0.597	0.631	0.624	0.599	0.697	0.650	0.359	0.360
	P(W/D)	0.182	0.173	0.174	0.160	0.186	0.413	0.382	0.422	0.401	0.319	0.186	0.142
	ALPHA	0.622	0.634	0.662	0.611	0.601	0.679	0.707	0.635	0.631	0.548	0.548	0.562
	BETA	0.553	0.577	0.513	0.735	1.091	0.914	0.559	0.657	0.799	1.027	0.680	0.533
TALLAHASSEE,FL	P(W/W)	0.387	0.433	0.404	0.379	0.483	0.573	0.633	0.577	0.500	0.437	0.344	0.387
	P(W/D)	0.241	0.286	0.225	0.187	0.206	0.304	0.496	0.329	0.254	0.110	0.163	0.219
	ALPHA	0.744	0.696	0.628	0.591	0.722	0.652	0.670	0.745	0.555	0.656	0.625	0.696
	BETA	0.583	0.830	0.973	0.901	0.628	0.836	0.727	0.665	1.288	0.903	0.768	0.780
TAMPA,FL	P(W/W)	0.309	0.409	0.397	0.370	0.359	0.568	0.602	0.583	0.553	0.438	0.327	0.267
	P(W/D)	0.180	0.201	0.169	0.118	0.169	0.270	0.436	0.474	0.350	0.178	0.132	0.181
	ALPHA	0.669	0.719	0.631	0.687	0.578	0.655	0.624	0.701	0.632	0.672	0.641	0.687
	BETA	0.526	0.634	0.951	0.621	0.758	0.713	0.811	0.668	0.718	0.490	0.646	0.497
ATLANTA,GA	P(W/W)	0.502	0.490	0.433	0.426	0.462	0.473	0.548	0.437	0.490	0.561	0.385	0.468
	P(W/D)	0.261	0.291	0.286	0.247	0.188	0.258	0.318	0.208	0.163	0.119	0.207	0.258
	ALPHA	0.718	0.727	0.689	0.723	0.726	0.765	0.681	0.711	0.661	0.622	0.668	0.743
	BETA	0.566	0.618	0.734	0.717	0.613	0.453	0.571	0.561	0.671	0.627	0.621	0.589

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
AUGUSTA,GA	P(W/W)	0.477	0.434	0.473	0.436	0.503	0.492	0.532	0.437	0.458	0.482	0.414	0.456
	P(W/D)	0.232	0.290	0.253	0.220	0.183	0.227	0.271	0.233	0.180	0.113	0.165	0.220
	ALPHA	0.733	0.787	0.689	0.637	0.754	0.813	0.614	0.641	0.694	0.643	0.618	0.738
	BETA	0.528	0.537	0.654	0.657	0.556	0.511	0.696	0.695	0.632	0.558	0.463	0.489
MACON,GA	P(W/W)	0.468	0.519	0.478	0.398	0.524	0.472	0.559	0.502	0.503	0.492	0.370	0.442
	P(W/D)	0.250	0.283	0.263	0.214	0.182	0.257	0.340	0.239	0.184	0.118	0.176	0.248
	ALPHA	0.701	0.789	0.666	0.632	0.597	0.637	0.692	0.751	0.623	0.594	0.734	0.756
	BETA	0.527	0.559	0.710	0.693	0.730	0.630	0.511	0.472	0.631	0.653	0.437	0.580
SAVANNAH,GA	P(W/W)	0.439	0.417	0.418	0.321	0.452	0.551	0.577	0.551	0.502	0.463	0.375	0.331
	P(W/D)	0.229	0.283	0.251	0.194	0.203	0.264	0.394	0.292	0.244	0.131	0.158	0.215
	ALPHA	0.737	0.718	0.710	0.712	0.626	0.689	0.671	0.653	0.622	0.582	0.600	0.795
	BETA	0.456	0.499	0.602	0.623	0.861	0.775	0.798	0.823	0.825	0.692	0.474	0.434
BOISE,ID	P(W/W)	0.595	0.559	0.459	0.406	0.476	0.464	0.250	0.353	0.370	0.389	0.534	0.543
	P(W/D)	0.317	0.235	0.223	0.211	0.196	0.150	0.053	0.063	0.083	0.152	0.213	0.271
	ALPHA	0.846	0.920	0.998	0.841	0.740	0.854	0.826	0.676	0.801	0.998	0.998	0.883
	BETA	0.148	0.115	0.101	0.180	0.211	0.176	0.113	0.202	0.159	0.115	0.139	0.128
POCATELLO,ID	P(W/W)	0.511	0.524	0.479	0.380	0.508	0.509	0.286	0.360	0.353	0.370	0.450	0.548
	P(W/D)	0.289	0.253	0.230	0.213	0.194	0.169	0.095	0.107	0.099	0.110	0.194	0.259
	ALPHA	0.949	0.998	0.998	0.998	0.794	0.824	0.850	0.706	0.836	0.884	0.987	0.992
	BETA	0.097	0.080	0.082	0.145	0.167	0.185	0.111	0.199	0.146	0.165	0.111	0.090
CHICAGO,IL	P(W/W)	0.430	0.430	0.485	0.559	0.441	0.458	0.437	0.357	0.455	0.456	0.460	0.483
	P(W/D)	0.291	0.285	0.330	0.332	0.293	0.288	0.270	0.202	0.214	0.193	0.236	0.274
	ALPHA	0.681	0.782	0.705	0.733	0.783	0.692	0.602	0.689	0.718	0.640	0.735	0.666
	BETA	0.251	0.206	0.287	0.424	0.357	0.548	0.735	0.652	0.500	0.537	0.325	0.280
EVANSVILLE,IN	P(W/W)	0.467	0.457	0.485	0.483	0.493	0.459	0.455	0.393	0.418	0.446	0.440	0.490
	P(W/D)	0.242	0.276	0.288	0.336	0.252	0.243	0.263	0.181	0.170	0.166	0.214	0.260
	ALPHA	0.673	0.725	0.622	0.669	0.697	0.676	0.743	0.654	0.629	0.659	0.707	0.648
	BETA	0.479	0.472	0.635	0.509	0.608	0.508	0.517	0.593	0.604	0.504	0.507	0.528
FORTWAYNE,IN	P(W/W)	0.496	0.463	0.552	0.535	0.502	0.493	0.439	0.393	0.424	0.434	0.461	0.498
	P(W/D)	0.326	0.309	0.359	0.389	0.305	0.253	0.297	0.217	0.238	0.202	0.277	0.313
	ALPHA	0.667	0.676	0.743	0.781	0.830	0.838	0.713	0.762	0.758	0.653	0.830	0.668
	BETA	0.280	0.294	0.275	0.346	0.385	0.435	0.489	0.467	0.359	0.525	0.313	0.279
INDIANAPOLIS,IN	P(W/W)	0.466	0.462	0.496	0.543	0.513	0.421	0.406	0.358	0.415	0.428	0.412	0.518
	P(W/D)	0.291	0.277	0.344	0.332	0.304	0.266	0.273	0.218	0.192	0.175	0.259	0.291
	ALPHA	0.630	0.692	0.688	0.749	0.845	0.671	0.746	0.753	0.646	0.689	0.733	0.669
	BETA	0.387	0.362	0.423	0.430	0.390	0.578	0.582	0.437	0.580	0.507	0.461	0.375

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
DES MOINES,IA	P(W/W)	0.391	0.397	0.490	0.466	0.455	0.489	0.367	0.393	0.444	0.389	0.403	0.384
	P(W/D)	0.205	0.212	0.255	0.317	0.286	0.295	0.257	0.252	0.238	0.183	0.141	0.200
	ALPHA	0.762	0.821	0.698	0.713	0.681	0.664	0.697	0.693	0.691	0.661	0.536	0.831
	BETA	0.157	0.172	0.302	0.370	0.572	0.567	0.530	0.552	0.499	0.409	0.468	0.149
DUBUQUE,IA	P(W/W)	0.411	0.396	0.483	0.472	0.478	0.475	0.405	0.395	0.422	0.475	0.391	0.444
	P(W/D)	0.234	0.212	0.269	0.326	0.301	0.286	0.298	0.219	0.237	0.184	0.173	0.248
	ALPHA	0.722	0.804	0.814	0.802	0.733	0.752	0.673	0.752	0.644	0.746	0.595	0.744
	BETA	0.227	0.201	0.344	0.484	0.558	0.564	0.674	0.680	0.795	0.528	0.633	0.266
DODGE CITY,KS	P(W/W)	0.287	0.305	0.397	0.402	0.484	0.482	0.421	0.441	0.442	0.425	0.411	0.384
	P(W/D)	0.109	0.138	0.150	0.157	0.233	0.213	0.247	0.209	0.144	0.096	0.074	0.103
	ALPHA	0.927	0.795	0.660	0.733	0.670	0.750	0.709	0.616	0.591	0.592	0.783	0.819
	BETA	0.096	0.138	0.296	0.294	0.500	0.453	0.485	0.454	0.521	0.500	0.191	0.129
TOPEDA,KS	P(W/W)	0.336	0.301	0.480	0.471	0.460	0.471	0.442	0.381	0.419	0.419	0.388	0.342
	P(W/D)	0.151	0.186	0.172	0.243	0.293	0.294	0.228	0.215	0.202	0.147	0.123	0.154
	ALPHA	0.773	0.708	0.748	0.626	0.780	0.720	0.695	0.652	0.755	0.695	0.592	0.894
	BETA	0.169	0.234	0.343	0.543	0.441	0.740	0.685	0.698	0.551	0.560	0.469	0.232
WICHITA,KS	P(W/W)	0.500	0.316	0.462	0.419	0.393	0.577	0.433	0.357	0.412	0.231	0.400	0.250
	P(W/D)	0.060	0.212	0.194	0.322	0.246	0.188	0.254	0.292	0.123	0.137	0.157	0.111
	ALPHA	0.621	0.734	0.524	0.551	0.690	0.786	0.640	0.989	0.724	0.998	0.609	0.858
	BETA	0.256	0.256	0.666	0.503	0.640	0.628	0.634	0.304	0.639	0.416	0.461	0.232
COVINGTON,KY	P(W/W)	0.492	0.477	0.487	0.561	0.561	0.467	0.393	0.418	0.397	0.416	0.480	0.515
	P(W/D)	0.326	0.332	0.380	0.343	0.265	0.259	0.283	0.211	0.196	0.197	0.289	0.309
	ALPHA	0.655	0.708	0.603	0.696	0.794	0.672	0.763	0.648	0.797	0.719	0.680	0.684
	BETA	0.405	0.378	0.501	0.388	0.411	0.527	0.589	0.458	0.420	0.400	0.403	0.356
LEXINGTON,KY	P(W/W)	0.496	0.489	0.502	0.520	0.500	0.526	0.430	0.394	0.441	0.400	0.459	0.478
	P(W/D)	0.317	0.345	0.356	0.353	0.292	0.273	0.312	0.245	0.176	0.194	0.267	0.321
	ALPHA	0.630	0.751	0.652	0.647	0.680	0.778	0.734	0.631	0.666	0.725	0.708	0.678
	BETA	0.464	0.396	0.577	0.478	0.535	0.507	0.560	0.603	0.587	0.363	0.462	0.447
LOUISVILLE,KY	P(W/W)	0.472	0.466	0.484	0.512	0.547	0.513	0.449	0.379	0.420	0.383	0.439	0.486
	P(W/D)	0.301	0.323	0.355	0.331	0.256	0.222	0.297	0.201	0.182	0.188	0.257	0.291
	ALPHA	0.662	0.709	0.645	0.664	0.723	0.680	0.743	0.692	0.648	0.752	0.628	0.653
	BETA	0.447	0.453	0.586	0.497	0.489	0.549	0.463	0.576	0.664	0.435	0.517	0.469
BATON ROUGE,LA	P(W/W)	0.381	0.466	0.398	0.376	0.506	0.531	0.560	0.482	0.416	0.376	0.305	0.464
	P(W/D)	0.251	0.267	0.220	0.182	0.180	0.194	0.363	0.279	0.219	0.121	0.180	0.255
	ALPHA	0.654	0.664	0.645	0.582	0.652	0.811	0.700	0.767	0.721	0.617	0.712	0.725
	BETA	0.684	0.832	0.739	1.311	0.804	0.452	0.712	0.568	0.580	0.836	0.742	0.706

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
NEW ORLEANS,LA	P(W/W)	0.409	0.458	0.404	0.343	0.439	0.483	0.576	0.536	0.495	0.433	0.369	0.449
	P(W/D)	0.253	0.279	0.227	0.197	0.191	0.258	0.368	0.329	0.237	0.130	0.168	0.274
	ALPHA	0.575	0.615	0.570	0.604	0.660	0.691	0.705	0.642	0.646	0.694	0.593	0.633
	BETA	0.865	0.903	0.871	0.935	0.870	0.641	0.684	0.670	0.846	0.571	0.825	0.803
SHREVEPORT,LA	P(W/W)	0.497	0.434	0.436	0.430	0.488	0.497	0.375	0.375	0.444	0.376	0.429	0.480
	P(W/D)	0.221	0.237	0.248	0.245	0.186	0.154	0.187	0.163	0.163	0.131	0.205	0.222
	ALPHA	0.625	0.699	0.729	0.665	0.668	0.578	0.607	0.527	0.663	0.713	0.652	0.645
	BETA	0.599	0.621	0.514	0.875	0.834	0.868	0.637	0.759	0.740	0.583	0.692	0.704
CARIBOU,ME	P(W/W)	0.516	0.518	0.539	0.508	0.531	0.472	0.500	0.508	0.473	0.498	0.573	0.527
	P(W/D)	0.409	0.368	0.315	0.318	0.332	0.376	0.424	0.367	0.361	0.316	0.389	0.379
	ALPHA	0.779	0.826	0.756	0.808	0.858	0.782	0.719	0.682	0.609	0.676	0.788	0.720
	BETA	0.192	0.217	0.227	0.264	0.248	0.318	0.385	0.438	0.487	0.400	0.304	0.280
PORTLAND,ME	P(W/W)	0.442	0.422	0.475	0.536	0.473	0.451	0.361	0.364	0.416	0.484	0.515	0.493
	P(W/D)	0.295	0.341	0.281	0.310	0.321	0.310	0.261	0.299	0.234	0.229	0.308	0.299
	ALPHA	0.765	0.672	0.716	0.717	0.714	0.651	0.724	0.708	0.631	0.603	0.691	0.670
	BETA	0.413	0.540	0.490	0.421	0.370	0.423	0.402	0.384	0.597	0.614	0.605	0.566
BALTIMORE,MD	P(W/W)	0.446	0.411	0.504	0.502	0.447	0.392	0.333	0.458	0.421	0.365	0.414	0.407
	P(W/D)	0.263	0.264	0.293	0.319	0.277	0.260	0.243	0.247	0.180	0.164	0.251	0.244
	ALPHA	0.791	0.791	0.713	0.698	0.707	0.631	0.592	0.617	0.530	0.698	0.653	0.737
	BETA	0.334	0.430	0.462	0.419	0.418	0.639	0.771	0.746	0.817	0.599	0.548	0.491
BOSTON,MA	P(W/W)	0.460	0.476	0.500	0.511	0.461	0.443	0.402	0.401	0.375	0.454	0.523	0.456
	P(W/D)	0.333	0.359	0.315	0.302	0.313	0.305	0.248	0.286	0.252	0.229	0.307	0.294
	ALPHA	0.689	0.618	0.662	0.720	0.670	0.680	0.663	0.582	0.562	0.607	0.601	0.679
	BETA	0.456	0.564	0.558	0.474	0.469	0.427	0.448	0.637	0.709	0.596	0.653	0.640
NANTUCKET,MA	P(W/W)	0.498	0.443	0.445	0.483	0.412	0.355	0.316	0.397	0.461	0.448	0.527	0.500
	P(W/D)	0.353	0.369	0.352	0.316	0.281	0.223	0.218	0.255	0.212	0.214	0.319	0.344
	ALPHA	0.763	0.697	0.723	0.699	0.652	0.660	0.636	0.644	0.571	0.665	0.660	0.718
	BETA	0.415	0.538	0.488	0.465	0.507	0.402	0.603	0.656	0.727	0.591	0.545	0.493
DETROIT,MI	P(W/W)	0.496	0.465	0.500	0.527	0.463	0.455	0.357	0.352	0.450	0.468	0.493	0.510
	P(W/D)	0.351	0.329	0.335	0.332	0.313	0.289	0.241	0.225	0.221	0.180	0.262	0.351
	ALPHA	0.695	0.775	0.772	0.741	0.684	0.776	0.713	0.704	0.778	0.672	0.743	0.651
	BETA	0.211	0.203	0.231	0.339	0.345	0.388	0.475	0.528	0.335	0.440	0.289	0.261
GRAND RAPIDS,MI	P(W/W)	0.661	0.510	0.554	0.534	0.469	0.408	0.391	0.382	0.438	0.476	0.578	0.624
	P(W/D)	0.362	0.392	0.352	0.333	0.278	0.288	0.252	0.218	0.276	0.230	0.295	0.373
	ALPHA	0.802	0.788	0.762	0.772	0.706	0.699	0.756	0.757	0.646	0.673	0.727	0.805
	BETA	0.153	0.157	0.228	0.373	0.379	0.514	0.438	0.462	0.508	0.451	0.322	0.173

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
DULUTH,MN	P(W/W)	0.528	0.463	0.474	0.498	0.546	0.506	0.439	0.444	0.509	0.524	0.559	0.574
	P(W/D)	0.291	0.272	0.269	0.291	0.342	0.324	0.307	0.324	0.298	0.212	0.239	0.296
	ALPHA	0.819	0.798	0.676	0.713	0.723	0.700	0.701	0.635	0.716	0.688	0.618	0.730
	BETA	0.114	0.113	0.225	0.320	0.349	0.478	0.487	0.511	0.395	0.330	0.266	0.159
MINNEAPOLIS,MN	P(W/W)	0.416	0.414	0.419	0.407	0.502	0.496	0.361	0.383	0.455	0.431	0.407	0.447
	P(W/D)	0.221	0.188	0.275	0.283	0.321	0.331	0.304	0.266	0.252	0.182	0.198	0.247
	ALPHA	0.826	0.730	0.670	0.785	0.751	0.760	0.627	0.732	0.771	0.642	0.675	0.826
	BETA	0.103	0.170	0.263	0.268	0.372	0.419	0.620	0.492	0.359	0.392	0.224	0.117
COLUMBIA,MO	P(W/W)	0.412	0.405	0.456	0.477	0.445	0.473	0.454	0.340	0.415	0.403	0.353	0.424
	P(W/D)	0.181	0.224	0.274	0.309	0.278	0.279	0.243	0.205	0.199	0.182	0.163	0.208
	ALPHA	0.643	0.712	0.695	0.816	0.803	0.677	0.706	0.662	0.612	0.585	0.735	0.750
	BETA	0.315	0.318	0.359	0.374	0.503	0.596	0.605	0.565	0.805	0.745	0.368	0.275
KANSAS CITY,MO.	P(W/W)	0.364	0.304	0.438	0.485	0.439	0.450	0.393	0.443	0.448	0.464	0.357	0.381
	P(W/D)	0.157	0.216	0.215	0.260	0.305	0.284	0.235	0.203	0.214	0.156	0.135	0.166
	ALPHA	0.727	0.713	0.682	0.754	0.687	0.786	0.672	0.646	0.662	0.695	0.553	0.859
	BETA	0.259	0.288	0.440	0.454	0.580	0.654	0.783	0.736	0.780	0.633	0.521	0.238
ST. LOUIS,MO	P(W/W)	0.405	0.384	0.477	0.487	0.476	0.487	0.438	0.375	0.426	0.387	0.440	0.453
	P(W/D)	0.195	0.254	0.276	0.328	0.273	0.243	0.224	0.201	0.190	0.184	0.193	0.218
	ALPHA	0.753	0.670	0.725	0.814	0.716	0.664	0.674	0.735	0.796	0.813	0.692	0.753
	BETA	0.281	0.392	0.363	0.392	0.473	0.642	0.623	0.437	0.434	0.417	0.431	0.306
JACKSON,MS	P(W/W)	0.516	0.454	0.458	0.364	0.539	0.450	0.451	0.394	0.429	0.386	0.389	0.488
	P(W/D)	0.262	0.287	0.258	0.267	0.170	0.205	0.289	0.246	0.174	0.126	0.217	0.267
	ALPHA	0.636	0.758	0.670	0.657	0.684	0.673	0.759	0.623	0.540	0.551	0.652	0.679
	BETA	0.605	0.573	0.720	0.840	0.775	0.598	0.516	0.630	0.843	0.749	0.661	0.725
MERIDIAN,MS	P(W/W)	0.411	0.441	0.414	0.399	0.434	0.412	0.456	0.402	0.429	0.436	0.317	0.439
	P(W/D)	0.260	0.281	0.244	0.224	0.174	0.199	0.290	0.243	0.171	0.108	0.195	0.252
	ALPHA	0.812	0.786	0.764	0.835	0.800	0.874	0.783	0.740	0.753	0.590	0.836	0.853
	BETA	0.530	0.622	0.766	0.702	0.599	0.523	0.571	0.569	0.583	0.815	0.543	0.698
BILLINGS,MT	P(W/W)	0.442	0.500	0.439	0.475	0.544	0.491	0.328	0.376	0.414	0.307	0.347	0.489
	P(W/D)	0.198	0.184	0.241	0.233	0.270	0.314	0.177	0.170	0.165	0.160	0.166	0.139
	ALPHA	0.998	0.998	0.845	0.775	0.740	0.728	0.662	0.743	0.753	0.768	0.732	0.911
	BETA	0.094	0.095	0.130	0.267	0.261	0.290	0.189	0.218	0.241	0.191	0.181	0.111
GREAT FALLS,MT	P(W/W)	0.526	0.490	0.478	0.490	0.523	0.564	0.383	0.457	0.428	0.393	0.453	0.481
	P(W/D)	0.210	0.211	0.207	0.245	0.269	0.297	0.177	0.162	0.169	0.129	0.156	0.178
	ALPHA	0.923	0.913	0.998	0.738	0.675	0.692	0.818	0.731	0.787	0.814	0.899	0.988
	BETA	0.113	0.111	0.104	0.193	0.339	0.356	0.201	0.223	0.184	0.151	0.132	0.092

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
HAVRE,MT	P(W/W)	0.503	0.481	0.317	0.449	0.457	0.500	0.433	0.424	0.433	0.273	0.394	0.453
	P(W/D)	0.189	0.162	0.169	0.183	0.237	0.308	0.154	0.152	0.163	0.138	0.130	0.144
	ALPHA	0.998	0.998	0.998	0.883	0.747	0.712	0.781	0.669	0.752	0.765	0.940	0.988
	BETA	0.062	0.061	0.065	0.179	0.226	0.292	0.271	0.274	0.206	0.166	0.095	0.065
HELENA,MT	P(W/W)	0.429	0.328	0.421	0.373	0.498	0.573	0.381	0.361	0.446	0.331	0.390	0.481
	P(W/D)	0.215	0.184	0.200	0.249	0.260	0.266	0.180	0.207	0.147	0.159	0.185	0.183
	ALPHA	0.998	0.982	0.843	0.805	0.726	0.891	0.883	0.804	0.844	0.802	0.866	0.988
	BETA	0.070	0.071	0.106	0.143	0.232	0.213	0.152	0.175	0.142	0.134	0.095	0.075
KALISPELL,MT	P(W/W)	0.658	0.567	0.539	0.429	0.518	0.563	0.310	0.510	0.525	0.540	0.525	0.570
	P(W/D)	0.383	0.309	0.249	0.250	0.264	0.310	0.145	0.164	0.197	0.217	0.322	0.431
	ALPHA	0.998	0.998	0.998	0.834	0.862	0.807	0.829	0.798	0.866	0.968	0.857	0.921
	BETA	0.100	0.083	0.073	0.135	0.185	0.248	0.186	0.219	0.163	0.114	0.132	0.110
MILES CITY,MT	P(W/W)	0.444	0.467	0.385	0.488	0.507	0.491	0.344	0.386	0.460	0.324	0.355	0.487
	P(W/D)	0.212	0.193	0.200	0.213	0.262	0.309	0.230	0.166	0.146	0.135	0.159	0.168
	ALPHA	0.998	0.988	0.958	0.869	0.741	0.744	0.666	0.699	0.787	0.848	0.861	0.998
	BETA	0.063	0.077	0.084	0.182	0.265	0.346	0.294	0.242	0.186	0.122	0.114	0.074
GRAND ISLAND,NE	P(W/W)	0.409	0.422	0.413	0.514	0.474	0.500	0.353	0.383	0.441	0.308	0.250	0.287
	P(W/D)	0.108	0.181	0.178	0.204	0.278	0.259	0.271	0.221	0.188	0.113	0.109	0.120
	ALPHA	0.841	0.795	0.745	0.645	0.724	0.745	0.668	0.647	0.650	0.885	0.780	0.676
	BETA	0.120	0.155	0.224	0.441	0.478	0.546	0.476	0.501	0.471	0.263	0.159	0.202
ELKO,NV	P(W/W)	0.467	0.533	0.420	0.476	0.532	0.547	0.310	0.354	0.250	0.338	0.486	0.489
	P(W/D)	0.224	0.216	0.212	0.163	0.176	0.130	0.095	0.091	0.083	0.080	0.146	0.220
	ALPHA	0.797	0.928	0.858	0.905	0.960	0.809	0.828	0.565	0.779	0.738	0.998	0.921
	BETA	0.164	0.091	0.108	0.115	0.117	0.188	0.114	0.310	0.131	0.193	0.124	0.134
LAS VEGAS,NV	P(W/W)	0.271	0.311	0.346	0.250	0.211	0.071	0.275	0.161	0.258	0.300	0.333	0.356
	P(W/D)	0.061	0.065	0.055	0.048	0.025	0.022	0.067	0.082	0.040	0.041	0.056	0.047
	ALPHA	0.808	0.921	0.802	0.749	0.727	0.669	0.672	0.543	0.629	0.799	0.605	0.826
	BETA	0.200	0.125	0.149	0.182	0.157	0.245	0.263	0.340	0.313	0.155	0.380	0.162
RENO,NV	P(W/W)	0.496	0.454	0.380	0.349	0.414	0.386	0.294	0.420	0.297	0.250	0.500	0.484
	P(W/D)	0.138	0.113	0.135	0.101	0.101	0.074	0.067	0.049	0.044	0.046	0.093	0.138
	ALPHA	0.728	0.748	0.838	0.721	0.663	0.942	0.998	0.900	0.960	0.701	0.813	0.718
	BETA	0.275	0.258	0.150	0.182	0.253	0.138	0.095	0.107	0.158	0.233	0.166	0.265
WINNEMUCCA,NV	P(W/W)	0.467	0.426	0.443	0.351	0.448	0.554	0.243	0.289	0.340	0.385	0.496	0.473
	P(W/D)	0.198	0.177	0.153	0.146	0.147	0.113	0.053	0.052	0.058	0.087	0.149	0.193
	ALPHA	0.928	0.961	0.998	0.786	0.899	0.718	0.787	0.759	0.783	0.761	0.998	0.930
	BETA	0.123	0.115	0.094	0.172	0.138	0.224	0.106	0.192	0.142	0.179	0.123	0.119

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
CONCORD, NH	P(W/W)	0.405	0.396	0.459	0.441	0.463	0.457	0.368	0.403	0.409	0.422	0.494	0.461
	P(W/D)	0.300	0.307	0.295	0.321	0.317	0.296	0.298	0.295	0.241	0.211	0.333	0.293
	ALPHA	0.774	0.800	0.809	0.873	0.763	0.723	0.741	0.654	0.718	0.710	0.701	0.670
	BETA	0.314	0.347	0.320	0.317	0.322	0.385	0.407	0.469	0.507	0.516	0.473	0.475
MT WASHINGTON, NH	P(W/W)	0.648	0.673	0.724	0.710	0.632	0.628	0.616	0.638	0.634	0.646	0.726	0.714
	P(W/D)	0.524	0.569	0.441	0.436	0.397	0.439	0.473	0.416	0.409	0.313	0.495	0.537
	ALPHA	0.789	0.619	0.735	0.822	0.794	0.972	0.849	0.893	0.787	0.808	0.734	0.695
	BETA	0.390	0.727	0.463	0.391	0.470	0.407	0.494	0.520	0.536	0.551	0.551	0.561
ALBUQUERQUE, NM	P(W/W)	0.263	0.392	0.346	0.264	0.346	0.412	0.395	0.429	0.320	0.378	0.339	0.350
	P(W/D)	0.080	0.090	0.095	0.073	0.094	0.077	0.253	0.240	0.129	0.090	0.070	0.093
	ALPHA	0.840	0.898	0.964	0.712	0.699	0.718	0.744	0.804	0.836	0.739	0.998	0.858
	BETA	0.112	0.101	0.124	0.205	0.139	0.213	0.209	0.191	0.182	0.294	0.111	0.156
ALBANY, NY	P(W/W)	0.456	0.441	0.471	0.519	0.516	0.461	0.391	0.358	0.360	0.425	0.474	0.494
	P(W/D)	0.360	0.365	0.331	0.331	0.336	0.310	0.303	0.322	0.254	0.210	0.340	0.339
	ALPHA	0.755	0.683	0.747	0.708	0.673	0.741	0.695	0.705	0.672	0.709	0.788	0.673
	BETA	0.232	0.294	0.323	0.333	0.372	0.337	0.386	0.399	0.556	0.462	0.312	0.358
BUFFALO, NY	P(W/W)	0.704	0.658	0.613	0.595	0.483	0.397	0.363	0.446	0.480	0.555	0.630	0.699
	P(W/D)	0.578	0.485	0.421	0.409	0.339	0.276	0.283	0.300	0.270	0.239	0.412	0.533
	ALPHA	0.779	0.728	0.752	0.783	0.757	0.785	0.719	0.754	0.728	0.711	0.824	0.751
	BETA	0.188	0.203	0.236	0.270	0.316	0.307	0.419	0.461	0.407	0.374	0.287	0.205
NEW YORK, NY	P(W/W)	0.464	0.446	0.466	0.471	0.443	0.416	0.381	0.358	0.399	0.396	0.478	0.473
	P(W/D)	0.302	0.296	0.325	0.354	0.314	0.271	0.245	0.297	0.217	0.191	0.283	0.299
	ALPHA	0.739	0.671	0.683	0.650	0.664	0.765	0.627	0.583	0.667	0.608	0.683	0.658
	BETA	0.328	0.492	0.509	0.494	0.413	0.380	0.628	0.768	0.579	0.682	0.514	0.481
SYRACRUSE, NY	P(W/W)	0.655	0.657	0.631	0.583	0.510	0.413	0.445	0.399	0.467	0.532	0.608	0.674
	P(W/D)	0.494	0.487	0.415	0.388	0.350	0.301	0.284	0.308	0.262	0.266	0.425	0.561
	ALPHA	0.893	0.778	0.736	0.800	0.783	0.735	0.715	0.722	0.805	0.824	0.806	0.840
	BETA	0.161	0.222	0.244	0.267	0.280	0.378	0.417	0.479	0.325	0.324	0.256	0.186
NORTH PLATTE, NE	P(W/W)	0.292	0.377	0.344	0.448	0.498	0.453	0.377	0.314	0.435	0.351	0.309	0.268
	P(W/D)	0.126	0.151	0.167	0.179	0.255	0.273	0.270	0.227	0.154	0.117	0.108	0.112
	ALPHA	0.845	0.750	0.731	0.683	0.700	0.635	0.769	0.676	0.705	0.704	0.813	0.785
	BETA	0.094	0.137	0.190	0.343	0.466	0.640	0.401	0.388	0.408	0.282	0.131	0.126
SCOTTSBLUFF, NE	P(W/W)	0.326	0.396	0.390	0.474	0.555	0.529	0.335	0.323	0.446	0.363	0.286	0.354
	P(W/D)	0.122	0.133	0.192	0.189	0.269	0.312	0.240	0.171	0.147	0.112	0.112	0.129
	ALPHA	0.998	0.998	0.877	0.858	0.715	0.699	0.676	0.789	0.600	0.720	0.868	0.998
	BETA	0.069	0.065	0.114	0.196	0.343	0.398	0.334	0.184	0.279	0.233	0.100	0.084

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
ASHEVILLE,NC	P(W/W)	0.448	0.507	0.519	0.520	0.535	0.498	0.551	0.542	0.532	0.582	0.450	0.492
	P(W/D)	0.265	0.302	0.344	0.296	0.265	0.296	0.358	0.279	0.184	0.158	0.221	0.239
	ALPHA	0.690	0.786	0.700	0.670	0.772	0.779	0.818	0.676	0.628	0.672	0.670	0.645
	BETA	0.378	0.382	0.445	0.452	0.318	0.398	0.282	0.491	0.630	0.485	0.425	0.435
CHARLOTTE,NC	P(W/W)	0.463	0.498	0.515	0.465	0.472	0.371	0.495	0.490	0.345	0.523	0.361	0.395
	P(W/D)	0.235	0.287	0.272	0.245	0.205	0.283	0.289	0.223	0.161	0.133	0.189	0.246
	ALPHA	0.752	0.900	0.728	0.751	0.844	0.766	0.679	0.695	0.652	0.688	0.765	0.634
	BETA	0.487	0.438	0.546	0.514	0.392	0.503	0.498	0.614	0.741	0.524	0.480	0.602
GREENSBORO,NC	P(W/W)	0.435	0.500	0.516	0.442	0.502	0.495	0.519	0.539	0.476	0.479	0.436	0.434
	P(W/D)	0.255	0.281	0.264	0.279	0.232	0.266	0.301	0.244	0.167	0.158	0.199	0.202
	ALPHA	0.739	0.819	0.803	0.725	0.721	0.646	0.694	0.643	0.535	0.562	0.697	0.713
	BETA	0.459	0.437	0.426	0.465	0.389	0.628	0.500	0.647	0.768	0.769	0.450	0.581
RALEIGH,NC	P(W/W)	0.416	0.508	0.465	0.433	0.442	0.459	0.521	0.480	0.431	0.400	0.418	0.425
	P(W/D)	0.251	0.258	0.261	0.247	0.247	0.236	0.264	0.243	0.147	0.150	0.201	0.204
	ALPHA	0.722	0.808	0.873	0.844	0.797	0.732	0.770	0.620	0.729	0.722	0.755	0.850
	BETA	0.485	0.454	0.390	0.405	0.428	0.541	0.571	0.813	0.643	0.592	0.473	0.434
BISMARCK,ND	P(W/W)	0.354	0.393	0.372	0.477	0.480	0.519	0.412	0.330	0.344	0.363	0.445	0.437
	P(W/D)	0.227	0.188	0.205	0.187	0.261	0.328	0.249	0.277	0.200	0.112	0.139	0.197
	ALPHA	0.998	0.935	0.803	0.704	0.698	0.673	0.690	0.626	0.755	0.822	0.828	0.998
	BETA	0.066	0.074	0.100	0.250	0.328	0.422	0.336	0.321	0.226	0.158	0.108	0.062
WILLISTON,ND	P(W/W)	0.409	0.374	0.349	0.397	0.469	0.480	0.396	0.297	0.383	0.364	0.393	0.469
	P(W/D)	0.227	0.204	0.206	0.187	0.189	0.322	0.240	0.205	0.176	0.119	0.155	0.169
	ALPHA	0.998	0.998	0.998	0.731	0.728	0.689	0.644	0.644	0.664	0.733	0.998	0.998
	BETA	0.071	0.077	0.065	0.251	0.287	0.360	0.345	0.326	0.274	0.179	0.081	0.067
NEWARK,NJ	P(W/W)	0.437	0.398	0.470	0.463	0.473	0.407	0.448	0.432	0.426	0.378	0.450	0.461
	P(W/D)	0.300	0.313	0.316	0.330	0.297	0.278	0.254	0.260	0.211	0.189	0.299	0.292
	ALPHA	0.781	0.763	0.704	0.738	0.719	0.736	0.630	0.616	0.600	0.691	0.720	0.738
	BETA	0.311	0.419	0.501	0.434	0.397	0.377	0.604	0.681	0.659	0.584	0.449	0.421
CLEVELAND,OH	P(W/W)	0.598	0.606	0.583	0.584	0.506	0.436	0.395	0.384	0.429	0.505	0.613	0.626
	P(W/D)	0.470	0.452	0.432	0.404	0.319	0.290	0.292	0.267	0.252	0.244	0.352	0.419
	ALPHA	0.702	0.781	0.780	0.811	0.794	0.769	0.639	0.691	0.823	0.775	0.748	0.762
	BETA	0.219	0.179	0.235	0.302	0.331	0.379	0.520	0.455	0.348	0.324	0.267	0.197
COLUMBUS,OH	P(W/W)	0.504	0.480	0.516	0.545	0.500	0.463	0.391	0.350	0.418	0.423	0.509	0.502
	P(W/D)	0.339	0.359	0.384	0.360	0.328	0.276	0.323	0.230	0.216	0.205	0.288	0.329
	ALPHA	0.683	0.757	0.664	0.788	0.754	0.733	0.720	0.822	0.766	0.879	0.740	0.739
	BETA	0.325	0.263	0.359	0.358	0.423	0.489	0.543	0.418	0.377	0.252	0.309	0.267

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
TOLEDO,OH	P(W/W)	0.534	0.450	0.515	0.520	0.519	0.459	0.392	0.364	0.433	0.431	0.505	0.521
	P(W/D)	0.350	0.326	0.364	0.366	0.287	0.252	0.260	0.229	0.251	0.186	0.278	0.363
	ALPHA	0.656	0.752	0.724	0.745	0.802	0.763	0.716	0.737	0.755	0.674	0.758	0.640
	BETA	0.240	0.228	0.251	0.305	0.313	0.434	0.504	0.508	0.321	0.381	0.287	0.283
OKLAHOMA CITY,OK	P(W/W)	0.370	0.415	0.450	0.399	0.492	0.447	0.407	0.328	0.360	0.374	0.424	0.396
	P(W/D)	0.123	0.172	0.179	0.197	0.217	0.205	0.175	0.190	0.190	0.117	0.100	0.125
	ALPHA	0.703	0.744	0.669	0.660	0.632	0.664	0.707	0.696	0.608	0.638	0.616	0.644
	BETA	0.247	0.255	0.387	0.638	0.873	0.696	0.572	0.551	0.880	0.867	0.529	0.351
TULSA,OK	P(W/W)	0.404	0.438	0.414	0.461	0.483	0.413	0.422	0.326	0.399	0.427	0.392	0.422
	P(W/D)	0.146	0.184	0.205	0.231	0.260	0.217	0.186	0.171	0.193	0.133	0.146	0.165
	ALPHA	0.711	0.757	0.672	0.707	0.658	0.647	0.591	0.662	0.638	0.582	0.605	0.625
	BETA	0.301	0.307	0.494	0.630	0.662	0.757	0.909	0.662	0.816	0.912	0.547	0.382
BURNS,OR	P(W/W)	0.566	0.519	0.545	0.438	0.468	0.433	0.255	0.352	0.339	0.508	0.596	0.606
	P(W/D)	0.353	0.223	0.233	0.178	0.180	0.157	0.067	0.082	0.072	0.127	0.201	0.243
	ALPHA	0.810	0.890	0.998	0.927	0.986	[illegible]	0.868	0.792	0.657	0.738	0.998	0.897
	BETA	0.152	0.142	0.096	0.107	0.126	[illegible]	0.148	0.164	0.263	0.203	0.146	0.168
MEACHUM,OR	P(W/W)	0.737	0.729	0.713	0.663	0.610	0.556	0.299	0.536	0.521	0.633	0.721	0.716
	P(W/D)	0.484	0.331	0.311	0.291	0.270	0.216	0.080	0.100	0.129	0.194	0.298	0.371
	ALPHA	0.844	0.900	0.998	0.918	0.920	0.838	0.816	0.688	0.792	0.801	0.927	0.906
	BETA	0.279	0.232	0.184	0.210	0.192	0.224	0.172	0.269	0.304	0.307	0.272	0.281
MEDFORD,OR	P(W/W)	0.655	0.557	0.588	0.534	0.538	0.452	0.344	0.367	0.318	0.529	0.627	0.657
	P(W/D)	0.361	0.269	0.236	0.189	0.174	0.111	0.036	0.053	0.086	0.159	0.273	0.281
	ALPHA	0.703	0.608	0.876	0.946	0.791	0.985	0.579	0.998	0.724	0.678	0.692	0.654
	BETA	0.346	0.344	0.174	0.111	0.190	0.138	0.296	0.153	0.248	0.337	0.345	0.422
PENDLETON,OR	P(W/W)	0.571	0.535	0.485	0.434	0.452	0.364	0.232	0.391	0.383	0.462	0.521	0.551
	P(W/D)	0.353	0.247	0.250	0.249	0.179	0.163	0.067	0.078	0.108	0.174	0.275	0.369
	ALPHA	0.966	0.977	0.998	0.938	0.874	0.843	0.957	0.932	0.913	0.813	0.933	0.909
	BETA	0.134	0.111	0.100	0.108	0.163	0.145	0.096	0.111	0.149	0.156	0.139	0.119
SALEM,OR	P(W/W)	0.791	0.728	0.750	0.638	0.611	0.555	0.404	0.494	0.507	0.659	0.776	0.755
	P(W/D)	0.411	0.341	0.293	0.304	0.215	0.151	0.045	0.086	0.148	0.233	0.338	0.427
	ALPHA	0.866	0.763	0.964	0.867	0.998	0.776	0.826	0.829	0.722	0.866	0.833	0.827
	BETA	0.435	0.380	0.270	0.198	0.172	0.221	0.148	0.157	0.296	0.337	0.380	0.434
PORTLAND,OR	P(W/W)	0.802	0.697	0.726	0.634	0.619	0.561	0.386	0.585	0.497	0.684	0.775	0.752
	P(W/D)	0.425	0.357	0.344	0.309	0.236	0.188	0.071	0.082	0.172	0.232	0.324	0.443
	ALPHA	0.830	0.840	0.998	0.945	0.853	0.854	0.788	0.843	0.790	0.962	0.868	0.879
	BETA	0.369	0.303	0.211	0.177	0.196	0.207	0.144	0.224	0.239	0.268	0.343	0.352

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
SEXT.SUMMIT,OR	P(W/W)	0.774	0.689	0.712	0.604	0.602	0.476	0.212	0.426	0.489	0.632	0.719	0.745
	P(W/D)	0.373	0.312	0.300	0.230	0.179	0.126	0.044	0.053	0.101	0.174	0.276	0.286
	ALPHA	0.730	0.712	0.887	0.835	0.776	0.890	0.819	0.748	0.745	0.729	0.694	0.731
	BETA	0.533	0.429	0.262	0.223	0.277	0.191	0.173	0.264	0.305	0.445	0.577	0.559
ROSWELL,NM	P(W/W)	0.314	0.352	0.358	0.286	0.329	0.307	0.408	0.421	0.384	0.531	0.360	0.359
	P(W/D)	0.063	0.097	0.072	0.056	0.091	0.117	0.197	0.173	0.125	0.078	0.053	0.070
	ALPHA	0.830	0.858	0.810	0.740	0.641	0.612	0.664	0.683	0.593	0.596	0.768	0.779
	BETA	0.196	0.160	0.170	0.233	0.274	0.330	0.330	0.331	0.367	0.432	0.176	0.183
PHILADELPHIA,PA	P(W/W)	0.464	0.393	0.438	0.459	0.437	0.395	0.372	0.421	0.407	0.381	0.441	0.478
	P(W/D)	0.268	0.295	0.298	0.313	0.275	0.272	0.246	0.256	0.185	0.171	0.257	0.255
	ALPHA	0.749	0.757	0.811	0.759	0.760	0.585	0.664	0.668	0.613	0.577	0.735	0.673
	BETA	0.342	0.388	0.442	0.405	0.365	0.684	0.665	0.615	0.746	0.678	0.467	0.502
PITTSBURGH,PA	P(W/W)	0.596	0.606	0.582	0.526	0.516	0.486	0.400	0.360	0.391	0.443	0.565	0.608
	P(W/D)	0.443	0.414	0.451	0.393	0.311	0.304	0.317	0.267	0.219	0.255	0.328	0.451
	ALPHA	0.751	0.836	0.731	0.847	0.772	0.733	0.728	0.651	0.723	0.695	0.841	0.765
	BETA	0.225	0.197	0.303	0.312	0.369	0.429	0.465	0.530	0.402	0.357	0.215	0.188
PROVIDENCE,RI	P(W/W)	0.422	0.461	0.453	0.484	0.445	0.465	0.354	0.372	0.400	0.405	0.495	0.450
	P(W/D)	0.336	0.323	0.321	0.298	0.301	0.297	0.256	0.304	0.211	0.208	0.292	0.329
	ALPHA	0.650	0.637	0.657	0.658	0.670	0.650	0.655	0.589	0.636	0.590	0.626	0.645
	BETA	0.477	0.568	0.562	0.549	0.451	0.371	0.491	0.640	0.683	0.735	0.633	0.592
CHARLESTON,SC	P(W/W)	0.438	0.448	0.478	0.377	0.443	0.569	0.539	0.520	0.481	0.472	0.383	0.404
	P(W/D)	0.244	0.268	0.265	0.194	0.205	0.259	0.381	0.310	0.231	0.134	0.171	0.222
	ALPHA	0.702	0.760	0.707	0.710	0.628	0.603	0.710	0.677	0.758	0.576	0.657	0.678
	BETA	0.478	0.506	0.604	0.551	0.749	0.941	0.840	0.753	0.684	0.894	0.437	0.501
COLUMBIA,SC	P(W/W)	0.492	0.477	0.481	0.449	0.417	0.446	0.515	0.502	0.462	0.529	0.392	0.416
	P(W/D)	0.227	0.283	0.262	0.227	0.206	0.246	0.290	0.260	0.162	0.112	0.168	0.229
	ALPHA	0.649	0.731	0.758	0.674	0.758	0.812	0.672	0.637	0.559	0.578	0.723	0.737
	BETA	0.612	0.559	0.593	0.634	0.581	0.475	0.676	0.837	1.031	0.824	0.473	0.507
HURON,SD	P(W/W)	0.333	0.445	0.379	0.457	0.485	0.465	0.358	0.360	0.368	0.433	0.368	0.331
	P(W/D)	0.171	0.167	0.189	0.252	0.263	0.324	0.261	0.254	0.176	0.114	0.134	0.169
	ALPHA	0.998	0.707	0.712	0.682	0.616	0.652	0.664	0.615	0.705	0.611	0.699	0.761
	BETA	0.055	0.181	0.185	0.300	0.426	0.514	0.388	0.391	0.322	0.419	0.175	0.127
RAPID CITY,SD	P(W/W)	0.370	0.503	0.444	0.518	0.519	0.557	0.394	0.338	0.362	0.360	0.382	0.411
	P(W/D)	0.156	0.200	0.222	0.233	0.306	0.317	0.239	0.208	0.167	0.103	0.157	0.155
	ALPHA	0.998	0.988	0.815	0.776	0.674	0.713	0.622	0.757	0.709	0.782	0.830	0.998
	BETA	0.064	0.088	0.130	0.263	0.346	0.378	0.390	0.251	0.250	0.201	0.098	0.070

RAINFALL GENERATION PARAMETERS

STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
CHATTANOOGA,TN	P(W/W)	0.513	0.517	0.489	0.493	0.473	0.465	0.541	0.457	0.443	0.489	0.453	0.453
	P(W/D)	0.268	0.295	0.289	0.270	0.228	0.264	0.263	0.240	0.201	0.154	0.217	0.263
	ALPHA	0.727	0.769	0.671	0.719	0.794	0.721	0.801	0.679	0.632	0.738	0.784	0.718
	BETA	0.623	0.612	0.725	0.635	0.457	0.464	0.481	0.502	0.732	0.519	0.572	0.733
KNOXVILLE,TN	P(W/W)	0.506	0.531	0.506	0.528	0.473	0.444	0.494	0.422	0.477	0.503	0.467	0.466
	P(W/D)	0.317	0.329	0.327	0.294	0.253	0.291	0.304	0.257	0.191	0.170	0.271	0.289
	ALPHA	0.747	0.774	0.737	0.759	0.916	0.720	0.778	0.681	0.732	0.729	0.731	0.664
	BETA	0.489	0.517	0.550	0.442	0.336	0.528	0.452	0.493	0.478	0.439	0.490	0.644
MEMPHIS,TN	P(W/W)	0.472	0.455	0.491	0.431	0.482	0.469	0.395	0.397	0.424	0.324	0.374	0.439
	P(W/D)	0.246	0.294	0.270	0.295	0.184	0.200	0.241	0.205	0.154	0.146	0.222	0.259
	ALPHA	0.645	0.753	0.755	0.729	0.717	0.755	0.698	0.620	0.658	0.657	0.715	0.686
	BETA	0.713	0.619	0.605	0.784	0.806	0.535	0.652	0.763	0.777	0.595	0.604	0.684
NASHVILLE,TN	P(W/W)	0.484	0.521	0.500	0.476	0.485	0.516	0.422	0.386	0.462	0.408	0.399	0.493
	P(W/D)	0.274	0.299	0.280	0.323	0.248	0.238	0.272	0.214	0.174	0.161	0.249	0.280
	ALPHA	0.655	0.835	0.705	0.763	0.743	0.718	0.705	0.751	0.647	0.738	0.805	0.721
	BETA	0.616	0.488	0.652	0.512	0.553	0.533	0.524	0.489	0.679	0.456	0.438	0.568
ABILENE,TX	P(W/W)	0.333	0.402	0.318	0.453	0.459	0.491	0.357	0.303	0.415	0.337	0.388	0.392
	P(W/D)	0.102	0.135	0.111	0.148	0.178	0.115	0.097	0.136	0.149	0.136	0.116	0.089
	ALPHA	0.603	0.796	0.864	0.741	0.676	0.633	0.637	0.587	0.609	0.611	0.707	0.700
	BETA	0.425	0.249	0.241	0.565	0.730	0.811	0.858	0.675	0.707	0.663	0.436	0.295
AMARILLO,TX	P(W/W)	0.313	0.353	0.326	0.376	0.443	0.448	0.464	0.373	0.303	0.477	0.419	0.365
	P(W/D)	0.081	0.117	0.121	0.107	0.212	0.207	0.203	0.203	0.147	0.090	0.061	0.092
	ALPHA	0.654	0.748	0.748	0.687	0.575	0.582	0.615	0.639	0.572	0.664	0.834	0.645
	BETA	0.214	0.173	0.240	0.352	0.560	0.753	0.546	0.560	0.564	0.479	0.214	0.237
AUSTIN,TX	P(W/W)	0.444	0.479	0.393	0.397	0.418	0.478	0.312	0.430	0.445	0.390	0.438	0.479
	P(W/D)	0.174	0.205	0.179	0.190	0.197	0.117	0.101	0.115	0.172	0.143	0.146	0.144
	ALPHA	0.601	0.555	0.632	0.613	0.571	0.611	0.547	0.643	0.637	0.550	0.593	0.556
	BETA	0.366	0.644	0.368	0.688	0.841	0.993	0.701	0.653	0.805	1.048	0.534	0.554
BROWNSVILLE,TX	P(W/W)	0.459	0.485	0.413	0.380	0.433	0.527	0.387	0.484	0.540	0.420	0.440	0.492
	P(W/D)	0.148	0.158	0.097	0.087	0.094	0.107	0.093	0.138	0.226	0.160	0.138	0.134
	ALPHA	0.614	0.469	0.646	0.517	0.535	0.586	0.615	0.628	0.579	0.507	0.623	0.558
	BETA	0.324	0.529	0.205	0.636	0.978	0.698	0.382	0.601	0.904	1.074	0.382	0.311
CORPUS CHRISTI,TX	P(W/W)	0.456	0.482	0.327	0.309	0.408	0.422	0.371	0.448	0.529	0.438	0.438	0.431
	P(W/D)	0.171	0.165	0.138	0.130	0.153	0.130	0.104	0.113	0.218	0.142	0.136	0.141
	ALPHA	0.483	0.547	0.635	0.453	0.581	0.560	0.562	0.597	0.565	0.553	0.636	0.544
	BETA	0.435	0.484	0.238	0.882	0.775	0.961	0.585	0.991	1.029	0.862	0.419	0.459

RAINFALL GENERATION PARAMETERS

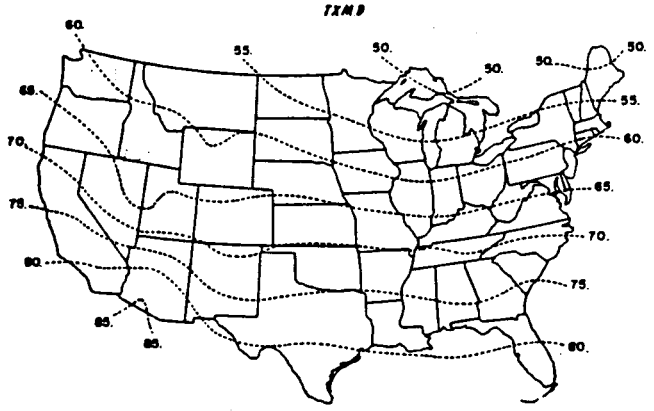
STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
DALLAS,TX	P(W/W)	0.431	0.456	0.404	0.471	0.447	0.392	0.348	0.298	0.440	0.320	0.447	0.435
	P(W/D)	0.153	0.195	0.203	0.196	0.202	0.146	0.105	0.139	0.143	0.126	0.134	0.131
	ALPHA	0.750	0.653	0.612	0.673	0.632	0.713	0.568	0.581	0.667	0.525	0.652	0.661
	BETA	0.358	0.448	0.586	0.924	0.824	0.659	0.703	0.655	0.882	1.281	0.704	0.589
ELPASO,TX	P(W/W)	0.368	0.352	0.288	0.235	0.257	0.250	0.320	0.441	0.376	0.328	0.320	0.368
	P(W/D)	0.060	0.067	0.075	0.046	0.043	0.087	0.229	0.168	0.093	0.077	0.064	0.080
	ALPHA	0.988	0.911	0.817	0.658	0.709	0.694	0.645	0.716	0.558	0.827	0.890	0.976
	BETA	0.107	0.179	0.165	0.188	0.208	0.279	0.338	0.222	0.510	0.214	0.146	0.117
GALVESTON,TX	P(W/W)	0.383	0.436	0.341	0.311	0.407	0.468	0.453	0.462	0.574	0.424	0.425	0.436
	P(W/D)	0.232	0.251	0.186	0.172	0.141	0.141	0.162	0.206	0.175	0.131	0.156	0.221
	ALPHA	0.640	0.622	0.567	0.551	0.589	0.580	0.609	0.635	0.523	0.727	0.613	0.691
	BETA	0.509	0.566	0.513	0.806	0.838	1.029	0.853	0.805	1.357	0.629	0.718	0.651
HOUSTON,TX	P(W/W)	0:407	0.492	0.369	0.410	0.440	0.478	0.443	0.464	0.541	0.508	0.410	0.473
	P(W/D)	0.253	0.237	0.218	0.212	0.189	0.156	0.214	0.219	0.186	0.135	0.205	0.232
	ALPHA	0.558	0.564	0.507	0.485	0.565	0.585	0.594	0.581	0.645	0.545	0.584	0.626
	BETA	0.615	0.754	0.574	0.899	1.085	1.112	0.710	0.747	0.843	1.034	0.774	0.663
SAN ANTONIO,TX	P(W/W)	0.446	0.494	0.409	0.387	0.403	0.417	0.319	0.378	0.486	0.445	0.448	0.432
	P(W/D)	0.180	0.195	0.166	0.179	0.195	0.123	0.088	0.115	0.167	0.135	0.140	0.158
	ALPHA	0.521	0.604	0.502	0.545	0.592	0.562	0.495	0.566	0.689	0.600	0.577	0.606
	BETA	0.392	0.453	0.420	0.584	0.719	0.947	0.841	0.769	0.650	0.762	0.593	0.343
TEMPLE,TX	P(W/W)	0.507	0.451	0.399	0.477	0.448	0.407	0.333	0.365	0.448	0.421	0.547	0.482
	P(W/D)	0.149	0.213	0.176	0.178	0.193	0.133	0.079	0.118	0.161	0.125	0.127	0.151
	ALPHA	0.659	0.735	0.713	0.680	0.630	0.704	0.705	0.584	0.686	0.488	0.633	0.590
	BETA	0.428	0.454	0.360	0.663	0.816	0.677	0.593	0.831	0.695	1.308	0.616	0.563
WACO,TX	P(W/W)	0.397	0.424	0.417	0.414	0.429	0.416	0.344	0.386	0.455	0.337	0.425	0.414
	P(W/D)	0.148	0.210	0.166	0.203	0.188	0.138	0.072	0.111	0.138	0.123	0.142	0.133
	ALPHA	0.650	0.744	0.676	0.573	0.612	0.651	0.639	0.711	0.706	0.626	0.707	0.677
	BETA	0.415	0.387	0.446	0.838	1.014	0.698	0.493	0.546	0.776	0.858	0.580	0.470
MILFORD,UT	P(W/W)	0.364	0.400	0.497	0.442	0.412	0.403	0.344	0.392	0.313	0.408	0.364	0.441
	P(W/D)	0.151	0.200	0.156	0.153	0.099	0.079	0.119	0.147	0.100	0.078	0.111	0.131
	ALPHA	0.863	0.990	0.981	0.920	0.998	0.770	0.771	0.890	0.721	0.848	0.889	0.956
	BETA	0.122	0.105	0.133	0.159	0.133	0.185	0.154	0.112	0.267	0.175	0.169	0.112
SALT LAKE CITY,UT	P(W/W)	0.479	0.397	0.463	0.525	0.487	0.500	0.315	0.373	0.389	0.461	0.434	0.497
	P(W/D)	0.226	0.263	0.236	0.239	0.165	0.139	0.104	0.139	0.111	0.108	0.170	0.230
	ALPHA	0.854	0.881	0.911	0.799	0.853	0.734	0.635	0.638	0.696	0.702	0.821	0.879
	BETA	0.165	0.169	0.178	0.276	0.206	0.249	0.299	0.264	0.219	0.265	0.212	0.170

RAINFALL GENERATION PARAMETERS

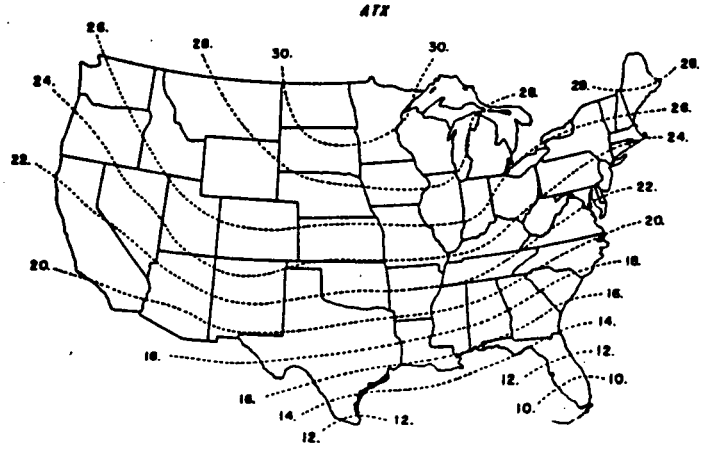
STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
NORFOLK,VA	P(W/W)	0.477	0.470	0.429	0.442	0.445	0.435	0.535	0.484	0.478	0.453	0.412	0.405
	P(W/D)	0.246	0.289	0.316	0.301	0.242	0.228	0.266	0.272	0.184	0.174	0.223	0.226
	ALPHA	0.728	0.757	0.744	0.782	0.727	0.645	0.704	0.608	0.519	0.619	0.666	0.823
	BETA	0.493	0.446	0.426	0.331	0.477	0.617	0.647	0.934	1.044	0.713	0.508	0.452
RICHMOND,VA	P(W/W)	0.474	0.446	0.460	0.477	0.490	0.472	0.448	0.472	0.424	0.382	0.396	0.402
	P(W/D)	0.252	0.266	0.284	0.253	0.243	0.226	0.271	0.249	0.187	0.172	0.237	0.215
	ALPHA	0.770	0.843	0.816	0.825	0.734	0.646	0.642	0.607	0.642	0.623	0.620	0.751
	BETA	0.374	0.431	0.393	0.353	0.419	0.641	0.802	0.881	0.661	0.743	0.642	0.553
OLYMPIA,WA	P(W/W)	0.816	0.766	0.758	0.698	0.586	0.542	0.489	0.571	0.601	0.707	0.787	0.788
	P(W/D)	0.452	0.344	0.321	0.276	0.185	0.194	0.079	0.106	0.160	0.267	0.349	0.455
	ALPHA	0.848	0.862	0.998	0.917	0.998	0.796	0.998	0.753	0.848	0.863	0.800	0.851
	BETA	0.482	0.392	0.264	0.236	0.171	0.194	0.149	0.262	0.289	0.419	0.530	0.462
SPOKANE,WA	P(W/W)	0.648	0.600	0.542	0.409	0.469	0.400	0.240	0.388	0.395	0.479	0.584	0.621
	P(W/D)	0.361	0.269	0.239	0.225	0.202	0.200	0.099	0.121	0.154	0.184	0.278	0.386
	ALPHA	0.955	0.998	0.956	0.933	0.889	0.702	0.878	0.746	0.824	0.910	0.903	0.887
	BETA	0.181	0.143	0.139	0.135	0.161	0.242	0.131	0.173	0.135	0.168	0.199	0.178
STAMPEDE PASS,WA	P(W/W)	0.867	0.822	0.807	0.774	0.684	0.714	0.530	0.649	0.638	0.723	0.807	0.858
	P(W/D)	0.457	0.418	0.388	0.379	0.323	0.284	0.161	0.209	0.251	0.330	0.361	0.442
	ALPHA	0.858	0.772	0.889	0.809	0.846	0.785	0.822	0.775	0.701	0.874	0.824	0.797
	BETA	0.698	0.680	0.486	0.458	0.285	0.307	0.217	0.282	0.543	0.596	0.763	0.775
YAKIMA,WA	P(W/W)	0.553	0.574	0.423	0.360	0.337	0.290	0.182	0.296	0.245	0.368	0.470	0.493
	P(W/D)	0.229	0.126	0.126	0.110	0.126	0.124	0.044	0.069	0.073	0.114	0.188	0.229
	ALPHA	0.811	0.873	0.998	0.988	0.977	0.807	0.902	0.998	0.872	0.878	0.974	0.809
	BETA	0.175	0.134	0.118	0.119	0.105	0.177	0.106	0.093	0.127	0.117	0.131	0.161
WALLA WALLA,WA	P(W/W)	0.592	0.560	0.486	0.457	0.451	0.336	0.306	0.328	0.415	0.454	0.539	0.548
	P(W/D)	0.377	0.262	0.259	0.240	0.197	0.181	0.054	0.085	0.119	0.200	0.304	0.370
	ALPHA	0.878	0.880	0.897	0.878	0.766	0.780	0.671	0.778	0.860	0.702	0.855	0.822
	BETA	0.174	0.146	0.148	0.167	0.229	0.197	0.208	0.201	0.196	0.235	0.180	0.174
CHARLESTON,WV	P(W/W)	0.541	0.551	0.577	0.548	0.550	0.500	0.466	0.473	0.473	0.464	0.514	0.521
	P(W/D)	0.383	0.395	0.397	0.395	0.314	0.264	0.369	0.249	0.213	0.222	0.279	0.384
	ALPHA	0.741	0.730	0.761	0.828	0.747	0.827	0.680	0.683	0.780	0.693	0.850	0.746
	BETA	0.315	0.344	0.364	0.304	0.365	0.351	0.598	0.547	0.398	0.380	0.289	0.300
GREEN BAY,WI	P(W/W)	0.400	0.393	0.495	0.493	0.471	0.487	0.398	0.405	0.426	0.467	0.425	0.420
	P(W/D)	0.282	0.217	0.262	0.271	0.339	0.298	0.273	0.267	0.293	0.196	0.223	0.286
	ALPHA	0.821	0.822	0.808	0.781	0.718	0.734	0.688	0.787	0.728	0.724	0.754	0.825
	BETA	0.130	0.159	0.180	0.346	0.362	0.407	0.509	0.342	0.454	0.365	0.252	0.150

RAINFALL GENERATION PARAMETERS

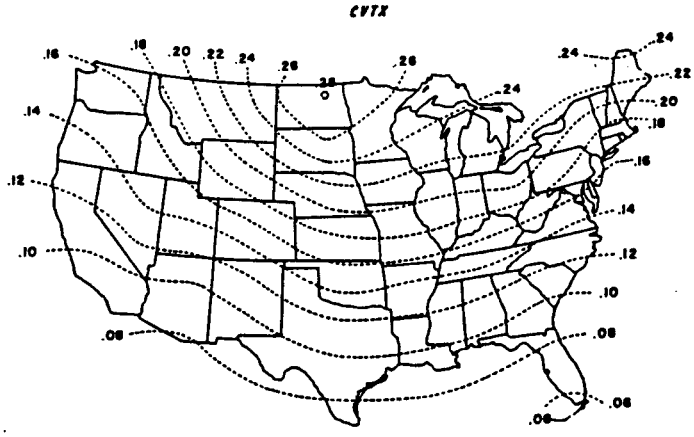
STATION		JAN	FEB	MAR	APR	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
LACROSSE,WI	P(W/W)	0.320	0.410	0.425	0.406	0.515	0.448	0.359	0.412	0.465	0.403	0.414	0.413
	P(W/D)	0.233	0.161	0.272	0.274	0.296	0.306	0.267	0.245	0.242	0.204	0.178	0.221
	ALPHA	0.838	0.778	0.723	0.791	0.862	0.728	0.732	0.816	0.722	0.783	0.662	0.874
	BETA	0.127	0.188	0.264	0.362	0.356	0.554	0.555	0.437	0.516	0.345	0.310	0.131
MADISON,WI	P(W/W)	0.392	0.408	0.468	0.487	0.522	0.452	0.380	0.369	0.432	0.471	0.419	0.455
	P(W/D)	0.284	0.204	0.292	0.322	0.287	0.297	0.282	0.256	0.245	0.204	0.219	0.218
	ALPHA	0.794	0.751	0.783	0.709	0.713	0.695	0.655	0.689	0.631	0.688	0.654	0.767
	BETA	0.137	0.170	0.220	0.350	0.408	0.568	0.616	0.544	0.549	0.413	0.329	0.214
MILWAUKEE,WI	P(W/W)	0.481	0.449	0.466	0.506	0.463	0.509	0.398	0.410	0.464	0.475	0.414	0.466
	P(W/D)	0.288	0.260	0.299	0.349	0.313	0.285	0.288	0.226	0.240	0.206	0.243	0.269
	ALPHA	0.661	0.756	0.711	0.758	0.800	0.670	0.635	0.650	0.638	0.670	0.692	0.695
	BETA	0.208	0.167	0.281	0.323	0.297	0.486	0.584	0.525	0.472	0.390	0.323	0.239
CHEYENNE,WY	P(W/W)	0.360	0.414	0.489	0.527	0.597	0.488	0.425	0.373	0.444	0.386	0.398	0.343
	P(W/D)	0.125	0.176	0.225	0.206	0.251	0.282	0.293	0.255	0.159	0.123	0.133	0.131
	ALPHA	0.998	0.924	0.833	0.864	0.749	0.689	0.742	0.737	0.735	0.794	0.942	0.967
	BETA	0.064	0.071	0.117	0.159	0.283	0.302	0.218	0.222	0.214	0.191	0.091	0.065



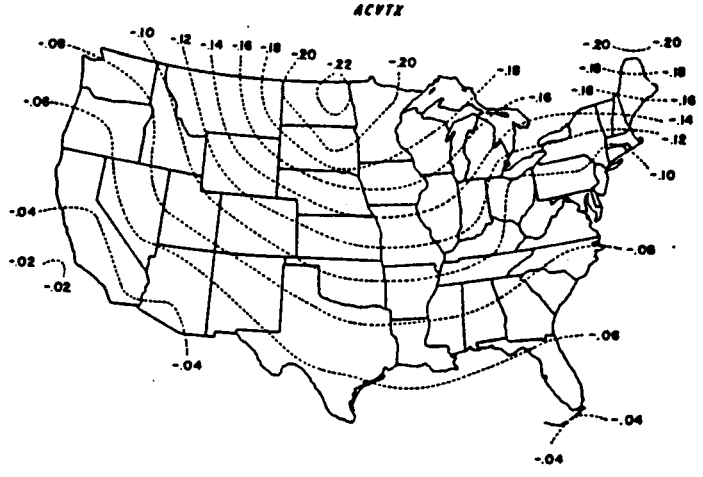
Distribution of the mean of t_{max} for dry days ($\overline{TXm}$), °F.



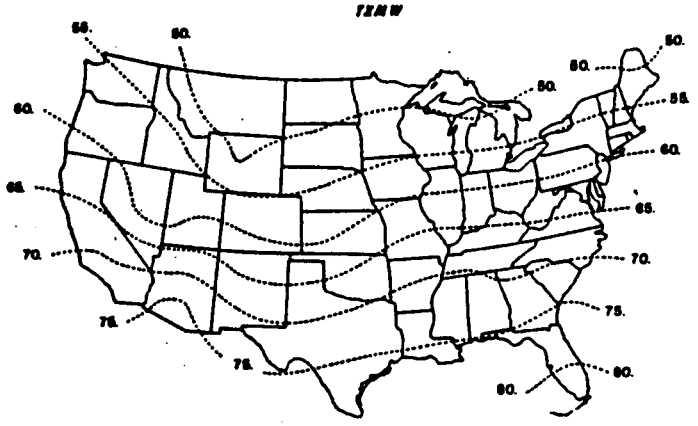
Distribution of the amplitude to t_{max} for wet or dry days (ATX), °F.



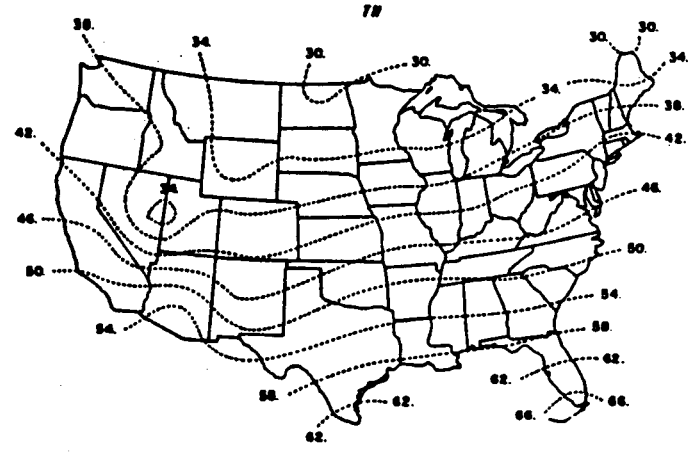
Distribution of the mean of the coefficient of variation of t_{max} for wet or dry days (CVTX).



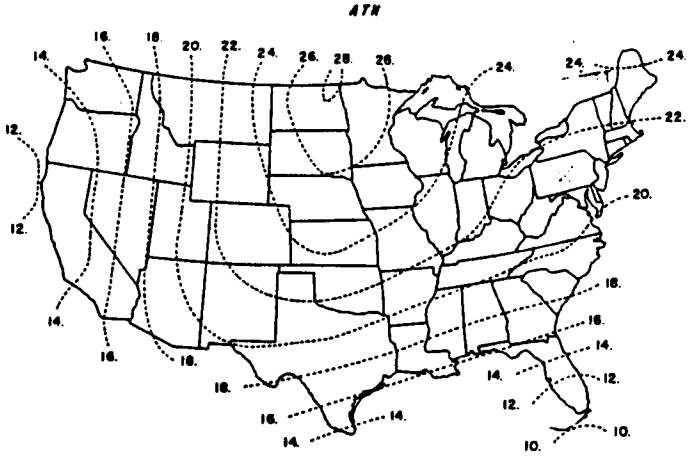
Distribution of the amplitude of the coefficient of variation of t_{max} for wet or dry days (ACVTX).



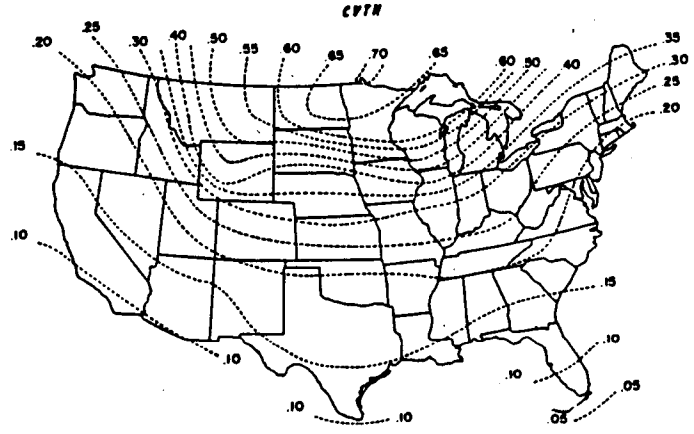
Distribution of the mean of t_{max} for wet days (TXMW), °F.



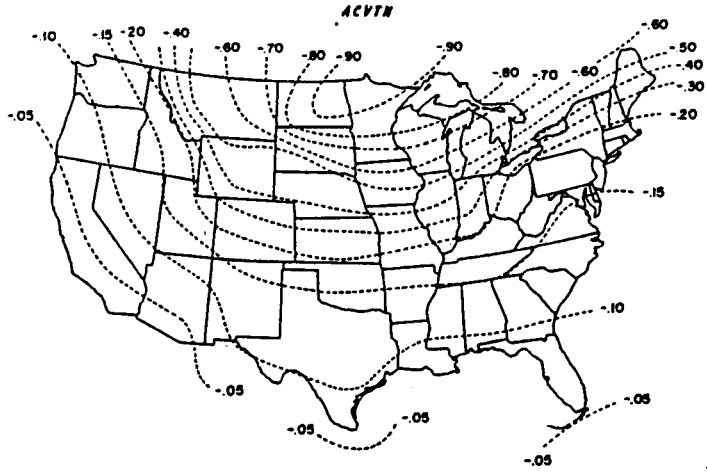
Distribution of the mean of t_{min} for wet or dry days (TN), °F.



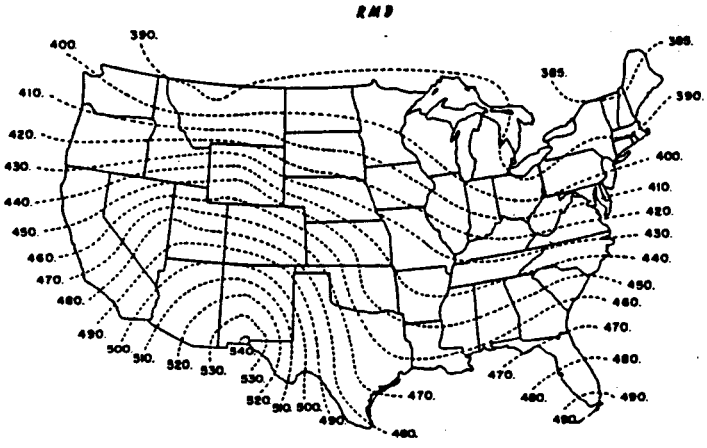
Distribution of the amplitude ot t_{min} for wet or dry days (ATN), °F.



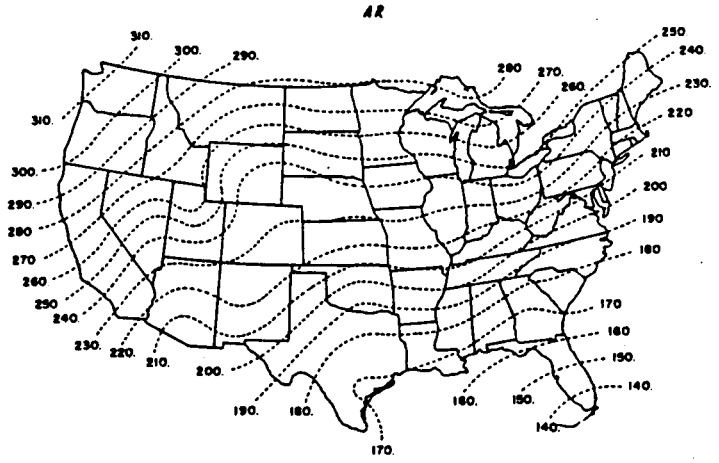
Distribution of the mean of the coefficient of variation of t_{min} for wet or dry days (CVTN).



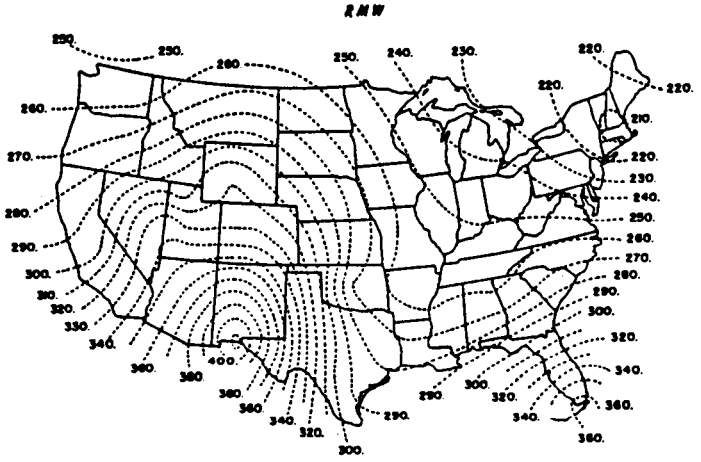
Distribution of the amplitude of the coefficient of variation of t_{min} for wet or dry days (ACVTN).



Distribution of the mean of r for dry days (RMD), 1y.



Distribution of the amplitude of r for dry days (AR), ly.



Distribution of the mean of r for wet days (RMW), ly.

USER'S GUIDE APPENDIX C

DESCRIPTIONS OF QC ERRORS AND ACTIONS

Julian date errors:

2 Missing one or more daily summaries prior to this date.
  Actions: 1 Date changed
  2 Data for missing day(s) filled in

3 The date for this data is not valid.
  Actions: 1 Date changed
  2 Data deleted

4 The date for this day's data is out of sequence.
  Actions: 1 Date changed
  2 Data deleted

5 Data for this date was missing.
  Actions: 1 Data like last day of data in current file
  2 Data like first day after missing data
  3 Data averaged from these two days

All other QC errors have the same following actions:
1 Value left unchanged
2 Value made equal to previous day's value
3 New value entered by user

6 Max. temperature too high or too low.
7 Max. temperature constant over time.
8 Max. temperature change over one day is greater than allowable.
9 Max. temperature is less than min. temperature.
10 Max. temperature was edited by user.

11 Min. temperature too high or too low.
12 Min. temperature constant over time.
13 Min. temperature change over one day is greater than allowable.
14 Min. temperature was edited by user.

15 Total solar radiation too high or too low.
16 Total solar radiation was edited by user.

17 Total precipitation too high or too low.
18 Total precipitation was edited by user.

PROGRAMMER'S GUIDE FOR
THE WEATHER DATA MANAGER PROGRAM

written by

William S. Donaldson

```
void AddDayToMissingDate(int *missingday_jdate, int *missingday_year,
                int *presentday_year)
/*-------------------------------------------------------------------------*/
/* Adds one day to the previous day's date to get the date for the day     */
/*  which was missing and is being filled with other data; if the date     */
/*  begins a new year, the program info and year of present day being run  */
/*  through QC are incremented, if necessary.                              */
/* Receives: *missingday_jdate:  address of the previous day's date; value */
/*                               is changed to become missing day's date   */
/*           *missingday_year:  address of the previous day's year; value  */
/*                              is changed to become missing day's year    */
/*           *presentday_year:  address of the year of the present data    */
/*                              set being run through QC                   */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: SaveProgramInfo                                                  */
/*-------------------------------------------------------------------------*/


int AllGeneratedDataStartDay(void)
/*-------------------------------------------------------------------------*/
/* Asks user for a starting date for generated data in the cur/gen'd file. */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: Julian day to start gen'd data.                                */
/* Calls: JulDate                                                          */
/*-------------------------------------------------------------------------*/
int month; /* month number (1-12) */
int day; /* day of month */
char month_day[8]; /* month and day entered by user */
int first_gen_day; /* first day of gen'd data to be put in current/gen'd file */


int AskToUseAllGeneratedData(void)
/*-------------------------------------------------------------------------*/
/* Asks user if a file of only generated data should be created.           */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: TRUE:  if file of gen'd data is to be created                  */
/*          FALSE:  if user wants to return to main menu                   */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int choice; /* user's choice */


void AveragePresentAndPreviousData(struct day *missingday_ptr,
                struct day *presentday_ptr, struct day *previousday_ptr)
/*-------------------------------------------------------------------------*/
/* Averages the max/min temp, SR, and precip of the day before missing     */
/*  data begins and of the present day's data set being run through QC;    */
/*  this is done when the user wants to make missing data like the average */
/*  of these two days' data                                                */
/* Receives: *missingday_ptr:  address of missing daily summary data       */
/*           *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


int CalDate(int jdate_to_convert, int year_to_convert, int *calendar_month,
                int *calendar_day)
/*-------------------------------------------------------------------------*/
```

```
/* Converts a julian date to a calendar date.                              */
/* Receives:  jdate_to_convert:  Julian date to convert                    */
/*            year_to_convert:  year of the Julian date to convert         */
/*            *calendar_month:  address to receive the calendar month      */
/*            *calendar_day:  address to receive the calendar day          */
/* Globals:  none.                                                         */
/* Returns:  CD_SUCCESS if date successfully converted                     */
/*           CD_INVALID_JDATE if julian date is invalid                    */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
static int days_per_month[12] = { 31, 28, 31, 30, 31, 30,

31, 31, 30, 31, 30, 31 };
int days_per_year = 365;


void CallStation()
/*-------------------------------------------------------------------------*/
/* Calls the Campbell 'telcom' program which retrieves data from the       */
/*  weather station datalogger and stores it in a comma delineated ASCII   */
/*  file.                                                                  */
/* Receives: none.                                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


int ChangeDateOrDeleteData(void)
/*-------------------------------------------------------------------------*/
/* Presents the user with the options to change the date of the data set of*/
/*  the present day being run through QC or to delete the data set (not    */
/*  save the data in the current file) and then returns the number of the  */
/*  user's choice.                                                         */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: 1:  if the user wants to change the date                       */
/*          2:  if the user wants to delete the data                       */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int ascii_number; /* ASCII value of the user's choice */
int choice;  /* user's choice */


int ChangeDateOrFillMissingData(struct day *previousday_ptr)
/*-------------------------------------------------------------------------*/
/* Presents the user with the options to change the date of the data set of*/
/*  the present day being run through QC or to fill-in data for the days   */
/*  which are missing, and then returns the number of the user's choice.   */
/* Receives: *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/* Globals: none.                                                          */
/* Returns: 1: if the user wants to change the date                        */
/*          2: if the user wants to fill-in the missing data               */
/* Calls: PrintDaysDataToScreen                                            */
/*-------------------------------------------------------------------------*/
int ascii_number; /* ASCII value of the user's choice */
int choice;  /* user's choice */


int CheckChange(float checkvalue, float previous1value, float maxchange)
/*-------------------------------------------------------------------------*/
/* Checks a value to see if it has changed more than is allowable.         */
/* Receives:  checkvalue:  value to check                                  */
/*            previous1value:  immediately previous value for comparison   */
/*            maxchange:  the maximum allowable change (float)             */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value is less than the allowable change           */
/*           FAIL if the value is greater                                  */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
```

```
int CheckConstant(float checkvalue, float previous1value, float previous2value)
/*-------------------------------------------------------------------------*/
/* Checks a value to see if it is the same as the past one or two values.  */
/* Receives:  checkvalue:  value to check                                  */
/*            previous1value:  immediately previous value for comparison   */
/*            previous2value:  next previous value for comparison          */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value is not the same as the past value(s)        */
/*           FAIL if the value is the same                                 */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/


int CheckExtreme(float checkvalue, float maxmin[MAXMIN])
/*-------------------------------------------------------------------------*/
/* Determines if a value is within a range from min to max.                */
/* Receives:  checkvalue:  value to check                                  */
/*            maxmin[]:  array with max and min values in it               */
/* Globals:  none.                                                         */
/* Returns:  PASS if value is within range                                 */
/*           FAIL if value is out of range                                 */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/


int CheckMaxMin(float checkvalue, float opposite_value)
/*-------------------------------------------------------------------------*/
/* Checks a value to see if it is > or < its opposite value.               */
/* Receives:  checkvalue:  value to check                                  */
/*            opposite_value:  the opposite value for comparison           */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value is > the opposite value                     */
/*           FAIL if the value is < the opposite value                     */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/


int ChooseHowToFillMissingData(struct day *presentday_ptr,
                struct day *previousday_ptr)
/*-------------------------------------------------------------------------*/
/* Presents the user with the options of how to fill-in the data for the   */
/*  missing day(s):  like the data of the day before the missing day(s)    */
/*  begin; like the data of the present day being run through QC; or like  */
/*  an average of these two days; then returns the number of the user's    */
/*  choice.                                                                */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/* Globals: month_name                                                     */
/* Returns: 1:  if the user wants to make the data like the previous day's */
/*          2:  if the user wants to make the data like the present day's  */
/*          3:  if the user wants to make the data like an ave. of the two */
/* Calls: CalDate                                                          */
/*        PrintDaysDataToScreen                                            */
/*-------------------------------------------------------------------------*/
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int previousday_month; /* month of the previous daily summary */
int previousday_day; /* day of the present daily summary */
int ascii_number; /* ASCII value of the user's choice */
int choice;  /* user's choice */


float ConvertForQC(float value_to_convert, int val_code)
/*-------------------------------------------------------------------------*/
/* Converts SI units to English units for temperature and precip.          */
/* Receives:  value_to_convert:  value to be converted to proper units for */
/*                               QC tests                                  */
/*            val_code:  number signifying what the value represents       */
/* Globals:  none.                                                         */
/* Returns:  the converted float value                                     */
```

```
/* Calls:  none.                                                          */
/*------------------------------------------------------------------------*/
float converted_value; /* value converted to proper units for QC tests */


void ConvertRawData(struct day *presentday_ptr)
/*------------------------------------------------------------------------*/
/* Converts ave. solar rad. reading to total daily solar rad. in MJ/m2,   */
/*  and subtracts one day from the Julian date.                           */
/* Receives: *presentday_ptr:  address of the daily summary data for the  */
/*                             present day                                */
/* Globals: info                                                          */
/* Returns: none.                                                         */
/* Calls: none.                                                           */
/*------------------------------------------------------------------------*/


int DayJulianDateQC(struct day *presentday_ptr, struct day *previousday_ptr,
               int *jdate_error, int *jdate_action, int *difference)
/*------------------------------------------------------------------------*/
/* Checks the Julian date to see if it is within the range of 1-366 and   */
/*    if it follows the previous day's date.                              */
/* Receives: *presentday_ptr:  address of the daily summary data for the  */
/*                             present day                                */
/*           *previousday_ptr:  address of the collected daily summary    */
/*                              data for the day previous to the present  */
/*                              day being run through QC                  */
/*           *jdate_error: address of the QC error from Julian date tests */
/*           *jdate_action: address of QC action from Julian date tests   */
/*           *difference: address of the difference between the present   */
/*                        and previous Julian dates                       */
/* Globals: info                                                          */
/* Returns: PASS for all but one instance                                 */
/*          FAIL only if the Julian day failed a qc test and the user does */
/*          not want the data to be put in IBSNAT file                    */
/* Calls: CalDate                                                         */
/*        ChangeDateOrDeleteData                                          */
/*        ChangeDateOrFillMissingData                                     */
/*        DecideOnFirstDaysData                                           */
/*        GetDifference                                                   */
/*        GetNewDateFromUser                                              */
/*        PrintJulianDateWarning                                          */
/*------------------------------------------------------------------------*/
int choice;   /* user's choice */
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int previousday_month; /* month of the previous daily summary */
int previousday_day; /* day of the present daily summary */
int warning = 0;  /* number indicating type of QC error found (if any) */
int first_time_through = TRUE; /* idicates the first time through Julian */
                                                                        /*
QC tests */
int DecideOnFirstDaysData(struct day *);
void PrintJulianDateWarning(int, struct day *, int, int, int, int, int);
int ChangeDateOrDeleteData(void);
int ChangeDateOrFillMissingData(struct day *);
int GetNewDateFromUser(int, int *);
int GetDifference(int, int, int);


int DayMoisQC(float checkvalue, struct qc *qc_ptr)
/*------------------------------------------------------------------------*/
/* Checks the datalogger moisture value.                                  */
/* Receives: checkvalue:  value to check                                  */
/*           *qc_ptr: address of QC values                                */
/* Globals: none.                                                         */
/* Returns: PASS if value passes all tests                                */
/*          FAIL if value fails any one test                              */
/* Calls: CheckExtreme                                                    */
/*------------------------------------------------------------------------*/


int DayPrecQC(float checkvalue, int checkvalue_month, struct qc *qcptr)
```

```
/*-------------------------------------------------------------------------*/
/* Checks a precipitation value to see if it is reasonable.                */
/* Receives:  checkvalue:  value to check                                  */
/*            checkvalue_month:  month the value was collected             */
/*            *qcptr:  address of qc values                                */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value passes all qc tests                         */
/*           FAIL if the value fails any one qc test                       */
/* Calls:  CheckExtreme                                                    */
/*-------------------------------------------------------------------------*/


int DayQC(struct day *presentday_ptr, struct day *previousday_ptr, struct qc
              *qc_ptr, int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION],
              FILE *current_fp)
/*-------------------------------------------------------------------------*/
/* Checks a set of daily data to see if it is reasonable                   */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           *qc_ptr:  address of the QC values                            */
/*           erroractions:  array of QC errors and actions                 */
/*           *current_fp: pointer to current data file                     */
/* Globals: none.                                                          */
/* Returns:  PASS: for all but instance listed below                       */
/*           FAIL: if the Julian day failed a qc test and the user does    */
/*                 not want the data to be put in IBSNAT file              */
/* Calls: CalDate                                                          */
/*        DayJulianDateQC                                                  */
/*        DayMoisQC                                                        */
/*        DayPrecQC                                                        */
/*        DayRHQC                                                          */
/*        DaySRQC                                                          */
/*        DayTempQC                                                        */
/*        DayVoltQC                                                        */
/*        DayWindQC                                                        */
/*        FigureProperYear                                                 */
/*        FillMissingData                                                  */
/*        FixError                                                         */
/*-------------------------------------------------------------------------*/
int warning; /* number indicating type of QC error found (if any) */
float temporary_value;  /* used to hold a value which failed QC so user may */
                                                          /*    change it */
float dummy_float = 0.0;  /* used in a function call when not all parameters */
                                                            /*    are needed */
int i; /* loop counter */
int j; /* loop counter */
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int difference; /* difference between the present day's Julian date and the */
                                          /*    previous day's Julian date */
int jdate_error = 0;  /* error code (if any) resulting from Julian date QC */
int jdate_action = 0; /* action code (if any) resulting from Julian date QC */
int DayJulianDateQC(struct day *, struct day *, int *, int *, int *);
int DayMoisQC(float, struct qc *);
int DayRHQC(float, struct qc *, float, float, int);
int DayVoltQC(float, struct qc *);
int DayWindQC(float, float);
int FigureProperYear(int);
void FillMissingData(struct day *, struct day *, int, FILE *);
int FixError(float *, float, int, int, struct qc *, float);


int DayRHQC(float checkvalue, struct qc *qc_ptr, float previous1value,
                              float opposite_value, int val_code)
/*-------------------------------------------------------------------------*/
/* Checks relative humidity values.                                        */
/* Receives: checkvalue:  value to check                                   */
/*           *qc_ptr: address of QC values                                 */
/*           previous1value:  immediately previous value for comparison    */
/*           opposite_value:  opposite value for comparison                */
```

```
/*          val_code:  number signifying what the value represents       */
/* Globals: none.                                                        */
/* Returns: PASS if value passes all tests                               */
/*          FAIL if value fails any one test                             */
/* Calls: CheckConstant                                                  */
/*        CheckExtreme                                                   */
/*        CheckMaxMin                                                    */
/*-----------------------------------------------------------------------*/


int DaySRQC(float checkvalue, int checkvalue_month, struct qc *qcptr)
/*-----------------------------------------------------------------------*/
/* Checks a solar radiation value to see if it is reasonable.            */
/* Receives:  checkvalue:  value to check                                */
/*            checkvalue_month:  month the value was collected           */
/*            *qcptr:  address of qc values                              */
/* Globals:  none.                                                       */
/* Returns:  PASS if the value passes all qc tests                       */
/*           FAIL if the value fails any one qc test                     */
/* Calls:  CheckExtreme                                                  */
/*-----------------------------------------------------------------------*/


int DayTempQC(float checkvalue, int checkvalue_month, struct qc *qcptr,
              float previous1value, float opposite_value, int val_code)
/*-----------------------------------------------------------------------*/
/* Checks a temperature value to see if it is reasonable.                */
/* Receives:  checkvalue:  value to check                                */
/*            checkvalue_month:  month the value was collected           */
/*            *qcptr:  address of qc values                              */
/*            previous1value:  immediately previous value for comparison */
/*            opposite_value:  opposite value for comparison             */
/*            val_code:  number signifying what the value represents     */
/* Globals:  none.                                                       */
/* Returns:  PASS if the value passes all qc tests                       */
/*           FAIL if the value fails any one qc test                     */
/* Calls:  CheckExtreme                                                  */
/*         CheckConstant                                                 */
/*         CheckChange                                                   */
/*         CheckMaxMin                                                   */
/*-----------------------------------------------------------------------*/


int DayVoltQC(float checkvalue, struct qc *qc_ptr)
/*-----------------------------------------------------------------------*/
/* Checks the datalogger battery voltage value.                          */
/* Receives: checkvalue:  value to check                                 */
/*           *qc_ptr: address of QC values                               */
/* Globals: none.                                                        */
/* Returns: PASS if value passes all tests                               */
/*          FAIL if value fails any one test                             */
/* Calls: CheckExtreme                                                   */
/*-----------------------------------------------------------------------*/


int DayWindQC(float checkvalue, float previous1value)
/*-----------------------------------------------------------------------*/
/* Checks the wind speed value.                                          */
/* Receives: checkvalue:  value to check                                 */
/*           previous1value: immediately previous value for comparison   */
/* Globals: none.                                                        */
/* Returns: PASS if value passes all tests                               */
/*          FAIL if value fails any one test                             */
/* Calls: CheckConstant                                                  */
/*-----------------------------------------------------------------------*/


int DecideOnFirstDaysData(struct day *presentday_ptr)
/*-----------------------------------------------------------------------*/
/* Lets user decide what to do with the first daily summary collected.   */
/* Receives: *presentday_ptr:  address of the daily summary data for the */
/*                             present day                               */
/* Globals: info                                                         */
```

```
/*          month_name                                                      */
/* Returns: PASS if user wants to keep data                                 */
/*          FAIL if user wants to delete data                               */
/* Calls: CalDate                                                           */
/*        PrintDaysDataToScreen                                             */
/*        SaveProgramInfo                                                   */
/*        GetNewFirstDateFromUser                                           */
/*--------------------------------------------------------------------------*/
int warning; /* number indicating type of QC error found (if any) */
int choice; /* user's choice */
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int GetNewFirstDateFromUser(int *);


void DisplaySumData(FILE *yearlysum_fp, int generated_year[YRS_GEN_DATA],
                int first_gen_day, int last_gen_day, int first_gen_day_year,
                int last_gen_day_year)
/*--------------------------------------------------------------------------*/
/* For each year of gen'd data, a summary is displayed to the user of the   */
/*  ave. temp. and total precip for the period over which gen'd data is     */
/*  required; the summaries are ranked by temp or precip (user's choice).   */
/* Receives: yearlysum_fp:  file pointer to WGEN yearly summary file        */
/*           generated_year[]:  array of the year numbers of the gen'd data*/
/*           first_gen_day:  first day of gen'd data                        */
/*           last_gen_day:  last day of gen'd data                          */
/*           first_gen_day_year:  year of first day of gen'd data           */
/*           last_gen_day_year:  year of last day of gen'd data             */
/* Globals: none.                                                           */
/* Returns: none.                                                           */
/* Calls: GetSumPrecTemp                                                    */
/*        Sort                                                              */
/*--------------------------------------------------------------------------*/
int choice; /* user's choice */
int i; /* loop counter */
float gen_precip[YRS_GEN_DATA]; /* total precip for each gen'd year */
float gen_temp[YRS_GEN_DATA]; /* ave. temp for each gen'd year */
void GetSumPrecTemp(FILE *, float *, float *, int, int, int, int);
void Sort(int, float [], float [], int []);


void EditCurrentDataFile(void)
/*--------------------------------------------------------------------------*/
/* Lets user change data stored in the current data file.                   */
/* Receives: none.                                                          */
/* Globals: none.                                                           */
/* Returns: none.                                                           */
/* Calls: EditDayOfCurrentData                                              */
/*        JulDate                                                           */
/*--------------------------------------------------------------------------*/
int month; /* month number (1-12) */
int day; /* day of month */
int year; /* two-digit year */
char date_line[11]; /* string to read new date entered by user */
int day_to_edit; /* Julian date of daily summary that user wants to edit */
void EditDayOfCurrentData(int, int, int, int);


void EditDayOfCurrentData(int day_to_edit, int month, int day, int year)
/*--------------------------------------------------------------------------*/
/* Lets user change a day's data.                                           */
/* Receives: day_to_edit:  Julian date of daily summary user wants to edit */
/*           month:  month number (1-12)                                    */
/*           day:  day of month                                             */
/*           year:  two-digit year                                          */
/* Globals: none.                                                           */
/* Returns: none.                                                           */
/* Calls: GetValueToChange                                                  */
/*        OpenFile                                                          */
/*        GetFormattedDay                                                   */
/*        PrintFormattedDay                                                 */
/*        GetQCValues                                                       */
/*        GetNewValue                                                       */
```

```
/*-------------------------------------------------------------------------*/
int open_result; /* signifies if error occurred during file opening */
int day_to_edit_found = FALSE; /* indicates if daily summary that user wants */
                                                                        /*
to edit has been located in the current */
                                                                        /*
data file */
FILE *current_fp; /* pointer to current data file */
FILE *temporary_fp; /* pointer to temporary data file */
char location_id[3]; /* string for location ID (for IBSNAT format) */
char station_id[3]; /* string for station ID (for IBSNAT format) */
float latitude; /* latitude of weather station (for IBSNAT format) */
float longitude; /* longitude of weather station (for IBSNAT format) */
float parfac; /* factor for converting MJ/M2 to PAR (for IBSNAT format) */
float pardat; /* flag indicating if PAR is included in weather file (for */
                                /*    IBSNAT format */
struct formatted_day current_daydata; /* IBSNAT-format daily summary from */

         /*    current file */
struct qc qcvalues; /* values for QC tests */
int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION]; /* array of QC */

                                                    /* errors and actions */
int value_to_change;/* user's choice of which daily summary value to change */
int GetValueToChange(struct formatted_day *, int, int, int);
float GetNewValue(int, int, int, struct qc *);


void EditProgramInfo(int edit_course)
/*-------------------------------------------------------------------------*/
/* Allows user to edit information used by the program.                    */
/* Receives: edit_course:  signifies course user will follow while         */
/*                         editing program info                            */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: InitializeProgramInfo                                            */
/*        SaveProgramInfo                                                  */
/*        PrintProgramInfo                                                 */
/*        OpenFile                                                         */
/*-------------------------------------------------------------------------*/
int choice; /* user's choice */
int finished = FALSE; /* indicates when user is finished editing */
char station_name_line[9]; /* used to read the station name from user */
char file_line[15]; /* used to get file names from user */
char id_line[5]; /* used to get ID's from user */
char latlon_line[9]; /* used to get latitude and longitude from user */
char year_line[5]; /* used to get year data collection started from user */
float newlatlon; /* new value for latitude or longitude */
int year; /* two-digit year */
FILE *curgen_fp; /* pointer to current/gen'd file */
int open_result; /* signifies if error occurred during file opening */
char year_string[3]; /* year data collection starts; used in file names */
char old_QC_file[13]; /* name of old QC values file; used to rename file */
void InitializeProgramInfo(void);
void PrintProgramInfo(void);


void EditQCValues(void)
/*-------------------------------------------------------------------------*/
/* Allows user to edit QC values file.                                     */
/* Receives: none.                                                         */
/* Globals: units_code                                                     */
/* Returns: none.                                                          */
/* Calls: GetQCValues                                                      */
/*        ZeroQCValues                                                     */
/*        EnterQCMaxmin                                                    */
/*        SaveQCValues                                                     */
/*-------------------------------------------------------------------------*/
int choice; /* user's choice */
int finished = FALSE; /* indicates when user is finished editing */
struct qc qcvalues; /* values for QC tests */
int i; /* loop counter */
char *qc_measurement[6] = {"temperature",
```

```
                                                                    "solar
radiation",
                                                                    "precipitation",
                                                                    "relative
humidity",
                                                                    "datalogger
voltage",
                                                                    "datalogger
moisture",};
                                    /* string array of the names of measurements that have QC
values */
/* Values to access labels in qc_measurement[] */
const int temp = 0;
const int sr = 1;
const int prec = 2;
const int rh = 3;
const int volt = 4;
const int mois = 5;
char entered_string[9]; /* used to get value entered by user */
float entered_value; /* numeric value from the string entered by user */
void ZeroQCValues(struct qc *);
void EnterQCMaxmin(float [MAXMIN], char *, char *);
void SaveQCValues(struct qc *);


void EnterQCMaxmin(float qc_data[MAXMIN], char *measurement,
                   char *measurement_units)
/*-------------------------------------------------------------------------*/
/* Lets user enter one max and one min value.                              */
/* Receives: qc_maxmin[]:  array with max and min values in it             */
/*           *measurement:  address of string containing name of current   */
/*                          measurement                                    */
/*           *measurement_units:  address of string containing units of    */
/*                                current QC measurement                   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
char *maxmin[MAXMIN] = {"maximum",    /* string array with words to identify */
                                                                  "minimum");   /*    max
and min values */
int i; /* loop counter */
char entered_string[9]; /* used to get value entered by user */
float entered_value; /* numeric value from the string entered by user */


int FigureProperYear(int previousday_jdate)
/*-------------------------------------------------------------------------*/
/* Figures the year for the present day's date                             */
/* Receives: previousday_jdate: Julian date of the previous day's data     */
/* Globals: info                                                           */
/* Returns: the year for the present day's date                            */
/* Calls: SaveProgramInfo                                                  */
/*-------------------------------------------------------------------------*/


void FileViewEdit(void)
/*-------------------------------------------------------------------------*/
/* Allows user to edit current and QC files and view current, current/     */
/*    gen'd, back-up, and QC files.                                        */
/* Receives: none.                                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: FileViewEditMenu                                                 */
/*        ShowFile                                                         */
/*        GetViewEditAction                                                */
/*        EditCurrentDataFile                                              */
/*        EditQCValues                                                     */
/*-------------------------------------------------------------------------*/
int choice; /* user's choice */
int view_or_edit;  /* indicates user's choice to view or edit a file */
void FileViewEditMenu(void);
char GetViewEditAction(void);
```

```
void ShowFile(char *);
void EditCurrentDataFile(void);
void EditQCValues(void);


void FileViewEditMenu(void)
/*-------------------------------------------------------------------------*/
/* Shows user choice of files to view and/or edit.                         */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


void FillMissingData(struct day *presentday_ptr, struct day *previousday_ptr,
                int difference, FILE *current_fp)
/*-------------------------------------------------------------------------*/
/* Fills in data for the date(s) for which data are missing                */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           difference: difference between the present day's Julian date  */
/*                       and the previous day's Julian date                */
/*           *current_fp:  pointer to the current data file                */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: FormatData                                                       */
/*        PrintFormattedDay                                                */
/*        AddDayToMissingDate                                              */
/*        AveragePresentAndPreviousData                                    */
/*        ChooseHowToFillMissingData                                       */
/*-------------------------------------------------------------------------*/
int i; /* loop counter */
int j; /* loop counter */
int k; /* loop counter */
int choice; /* user's choice */
int missingday_jdate; /* Julian date of missing daily summary */
int missingday_year; /* year of missing daily summary */
struct day missingday; /* data of missing daily summary */
struct formatted_day formatted_data; /* daily summary in IBSNAT format */
int missing_erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION];
int ChooseHowToFillMissingData(struct day *, struct day *);
void AddDayToMissingDate(int *, int *, int *);
void AveragePresentAndPreviousData(struct day *, struct day *, struct day *);


int FixError(float *value_to_fix, float previous1value, int val_code,
                int presentday_month, struct qc *qc_ptr, float opposite_value)
/*-------------------------------------------------------------------------*/
/* Lets user decide what to do with a value which has failed QC.           */
/* Receives: *value_to_fix:  address of value that failed QC               */
/*           previous1value:  immediately previous value for comparison    */
/*           val_code:  number signifying what the value represents        */
/*           presentday_month:  month of the present daily summary         */
/*           *qc_ptr: address of QC values                                 */
/*           opposite_value:  opposite value for comparison                */
/* Globals: none.                                                          */
/* Returns: VALUE_UNCHANGED:  if user wants to keep value unchanged        */
/*          VALUE_EQUALS_PREVIOUS:  if user wants value set equal to the   */
/*                                  previous day's value                   */
/*          NEW_VALUE_ENTERED:  if user entered a new value                */
/* Calls: QCNewValue                                                       */
/*-------------------------------------------------------------------------*/
float newvalue; /* new value entered by user for one which failed QC */
int ascii_number; /* ASCII value of user's choice */
int choice; /* user's choice */
int keeping_newvalue = FALSE; /* indicates if the user wants to keep or */
                                                                        /*
change new value which failed QC tests */
```

```
void FormatData(struct formatted_day *formatted_ptr,
                struct day *unformatted_ptr)
/*-------------------------------------------------------------------------*/
/* Converts daily data (deg. F, inches) to IBSNAT units (deg. C, cm), and  */
/*  puts the other info for the IBSNAT file in the converted struct.       */
/* Receives: *unformatted_ptr:  address of the daily data to be formatted  */
/*           *formatted_ptr:  address of the formatted data                */
/* Globals:  info                                                          */
/* Returns:  none.                                                         */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/


void FormatGenData(struct gen_data *gendata_ptr,
                struct formatted_day *formatted_ptr,
                int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION], int year)
/*-------------------------------------------------------------------------*/
/* Converts a daily summary in WGEN format to IBSNAT format.               */
/* Receives: *gendata_ptr:  address of a day of gen'd data                 */
/*           *formatted_ptr:  address of the formatted data                */
/*           erroractions[][]:  array of QC errors and actions             */
/*           year:  two-digit year                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int i; /* loop counter */
int j; /* loop counter */


int GetDataLineFromFile(FILE *fp, char line_from_file[100])
/*-------------------------------------------------------------------------*/
/* Gets line of data from a file as a string.                              */
/* Receives: *fp:  pointer to file                                         */
/*           line_from_file[]:  string to read a line of data from file    */
/* Globals: none.                                                          */
/* Returns: FILE_END:  if end of file is reached                           */
/*          OK:  if line of data was read                                  */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
const maxfgets = 99;     /* Maximum length of a data line that is read */


void GetDay(FILE *raw_fp, struct day *presentday_ptr)
/*-------------------------------------------------------------------------*/
/* Gets line of daily data from raw data file.                             */
/* Receives: *raw_fp:  pointer to raw data file                            */
/*           *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: ScanDay                                                          */
/*-------------------------------------------------------------------------*/


int GetDifference(int presentday_jdate, int previousday_jdate,
                int previousday_year)
/*-------------------------------------------------------------------------*/
/* Figures the difference between the present day's date and the previous  */
/*  day's date.                                                            */
/* Receives:  presentday_jdate:  Julian date of the present daily summary  */
/*            previousday_jdate:  Julian date of the previous daily summary*/
/*            previousday_year:  year of the previous daily summary        */
/* Globals: none.                                                          */
/* Returns: the difference                                                 */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int difference; /* difference between the present and previous Julian dates */


void GetInfileValue(float *wgen_input_value)
/*-------------------------------------------------------------------------*/
/* Gets a value for the WGEN input file from user.                         */
```

```
/* Receives: *wgen_input_value:  address of value for WGEN input file      */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
char wgen_input_string[12]; /* used to get wgen input value entered by user */
float entered_value; /* numeric value from the string entered by user */


int GetFormattedDay(FILE *fp, struct formatted_day *formatted_ptr,
                int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION])
/*-------------------------------------------------------------------------*/
/* Reads a line of data in the IBSNAT format from the current data file,   */
/*  plus the month and day and any codes for qc errors and actions.        */
/* Receives: *fp:  pointer to file                                         */
/*           *formatted_ptr:  address of the formatted data                */
/*           erroractions[][]:  array of qc errors and actions             */
/* Globals: none.                                                          */
/* Returns: OK if not at end of file                                       */
/*          FILE_END if reach end of file                                  */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int scan_result;  /* signifies if any error/actions are recorded */
int i;  /* loop counter */
int j;  /* loop counter */
char month_from_file[4];  /* month name read from IBSNAT-formatted file */
int day_from_file;  /* day read from IBSNAT-formatted file */
int qc_error_from_file;  /* a QC error read from IBSNAT-formatted file */
int qc_action_from_file;  /* a QC action read from IBSNAT-formatted file */


int GetGenDay(FILE *fp, struct gen_data *gendata_ptr)
/*-------------------------------------------------------------------------*/
/* Obtains a day of gen'd data from a file.                                */
/* Receives: *fp:  pointer to file                                         */
/*           *gendata_ptr:  address of a day of generated data             */
/* Globals: none.                                                          */
/* Returns: OK:  if data was read without problem                          */
/*          FILE_END:  if problem occured during read                      */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


void GetGeneratedData(void)
/*-------------------------------------------------------------------------*/
/* Gets years of generated data and allows the user to choose one of these */
/*  years to put with the current data or to put in a file by itself;      */
/*  the chosen generated data are then placed in the current/generated     */
/*  data file.                                                             */
/* Receives: none.                                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: AskToUseAllGeneratedData                                         */
/*        SetGeneratedStartingDay                                          */
/*        AllGeneratedDataStartDay                                         */
/*        RunGen                                                           */
/*        GetGenDay                                                        */
/*        FormatGenData                                                    */
/*        GetNewInfileData                                                 */
/*        MoveToChosenYear                                                 */
/*        DisplaySumData                                                   */
/*        OpenFile                                                         */
/*        GetFormattedDay                                                  */
/*        PrintFormattedDay                                                */
/*-------------------------------------------------------------------------*/
int choice; /* user's choice */
FILE *current_fp; /* pointer to current data file */
FILE *dailysum_fp; /* pointer to WGEN daily summary file */
FILE *yearlysum_fp; /* pointer to WGEN yearly summary file */
FILE *wgeninput_fp; /* pointer to WGEN input parameters file */
FILE *curgen_fp; /* pointer to current/gen'd data file */
int open_result; /* signifies if error occurred during file opening */
int first_gen_day; /* first day of gen'd data to be put in current/gen'd file */
```

```
int first_gen_day_year; /* year of the first day of gen'd data to be put in */
                                                        /*    the current/gen'd
file */
int last_gen_day; /* last day of gen'd data to be put in current/gen'd file */
int last_gen_day_year; /* year of the last day of gen'd data to be put in */
                                                    /*    the current/gen'd file */
char location_id[3]; /* string for location ID (for IBSNAT format) */
char station_id[3]; /* string for station ID (for IBSNAT format) */
float latitude; /* latitude of weather station (for IBSNAT format) */
float longitude; /* longitude of weather station (for IBSNAT format) */
float parfac; /* factor to convert MJ/m2 to PAR (for IBSNAT format) */
float pardat; /* signifyies if PAR is included in file (for IBSNAT format) */
int use_all_generated_data = FALSE; /* signifies if user wants only gen'd */

        /*    data in the current/gen'd file */
struct gen_data gendata; /* gen'd data for a day */
struct formatted_day formatted_data; /* daily summary in IBSNAT format */
int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION];

                /* array of QC errors and actions */
int generated_year[YRS_GEN_DATA]; /* array of the year numbers of gen'd data */
char chosen_year_string[5]; /* string to get the listed number of the gen'd */
                                                                /*    year
chosen by user */
int chosen_year; /* listed number of the gen'd year chosen by user */
long start_year_pos; /* location in WGEN daily summary file where the chosen */
                                                        /*    year of gen'd data begins
*/
int AskToUseAllGeneratedData(void);
void SetGeneratedStartingDay(int, int, int *, int *);
int AllGeneratedDataStartDay(void);
void RunGen(void);
void FormatGenData(struct gen_data *, struct formatted_day *,
                int [VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION], int);
void GetNewInfileData(void);
void MoveToChosenYear(FILE *, int);
void DisplaySumData(FILE *, int [YRS_GEN_DATA], int, int, int, int);


void GetHour(FILE *raw_fp, struct hour *presenthour_ptr)
/*-------------------------------------------------------------------------*/
/* Gets line of hourly data from raw data file                             */
/* Receives: *raw_fp:  pointer to raw data file                            */
/*           *presenthour_ptr: address of the data for the present hour    */
/*                             being run through QC                        */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: ScanHour                                                         */
/*-------------------------------------------------------------------------*/


int GetNewDateFromUser(int previousday_year, int *presentday_year)
/*-------------------------------------------------------------------------*/
/* Lets the user enter a new date.                                         */
/* Receives: previousday_year:  year of the previous daily summary         */
/*           *presentday_year:  address of the year of the present daily   */
/*                              summary                                    */
/* Globals: info                                                           */
/* Returns: julian date of the date entered                                */
/* Calls: JulDate                                                          */
/*        SaveProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int year; /* two-digit year */
char date_line[11]; /* string to read new date entered by user */
int new_julian_date; /* Julian date of the new date entered by user */
int choice; /* user's choice */


int GetNewFirstDateFromUser(int *firstday_year)
/*-------------------------------------------------------------------------*/
/* Gets a new date from the user for the first daily summary collected     */
```

```
/* Receives: *firstday_year:  address of the year for the first daily      */
/*                            summary                                      */
/* Globals: none.                                                          */
/* Returns: the julian date of the entered date                            */
/* Calls: JulDate                                                          */
/*-------------------------------------------------------------------------*/
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int year; /* two-digit year */
char date_line[11]; /* string to read new date entered by user */
int new_julian_date; /* Julian date of the new date entered by user */


void GetNewInfileData(void)
/*-------------------------------------------------------------------------*/
/* Gets input parameters to run WGEN from user.                            */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: GetInfileValue                                                   */
/*        LoadInfileData                                                   */
/*        InitializeInfileValues                                           */
/*        SaveInfileData                                                   */
/*-------------------------------------------------------------------------*/
int open_result; /* signifies if error occurred during file opening */
int finished = FALSE; /* signifies when user is finished editing */
char wgen_input_desc[81]; /* string to get comment line for WGEN input file */
                                                                    /*    from user */
int choice; /* user's choice */
int i; /* loop counter */
struct wgen_input wgen_input_data; /* data for the WGEN input file */
void GetInfileValue(float *);
int LoadInfileData(struct wgen_input*);
void InitializeInfileValues(struct wgen_input*);
void SaveInfileData(struct wgen_input *);


float GetNewValue(int val_code, int unit_code, int month, struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Prints the name and units of the value to enter, gets a float value     */
/*  from user, converts it to proper units for the qc tests, runs qc       */
/*  procedures on it, repeats this if value fails qc and user wants to     */
/*  reenter it                                                             */
/* Receives:  val_code:  number signifying what the value represents       */
/*            unit_code:  code signifying units for a value                */
/*            month:  month number (1-12)                                  */
/*            *qcptr:  address of the qc values                            */
/* Globals:  value_code                                                    */
/*           units_code                                                    */
/* Returns:  the float value entered                                       */
/* Calls:  ConvertForQC                                                    */
/*         QCNewValue                                                      */
/*-------------------------------------------------------------------------*/
float newvalue;  /* new value entered by user for one which failed QC */
float converted_newvalue; /* new value converted to proper units for qc tests */
int keeping_newvalue = FAIL;  /* indicates if user wants to keep or re-enter */
                                                                        /*
the new value that failed QC tests */
float ConvertForQC(float, int);


void GetPrecip(FILE *raw_fp, int *precip_time, float *precip_amount)
/*-------------------------------------------------------------------------*/
/* Gets line of precip data from raw data file                             */
/* Receives: *raw_fp:  pointer to raw data file                            */
/*           *precip_time:  address of the time of precip occurrance       */
/*           *precip_amount:  address of the amount of precip occurrance   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
```

```
int GetQCValues(struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Opens qc file (returns if file doesn't exist), gets qc data (returns if */
/*  insufficient data), closes file.                                       */
/* Receives:  *qcptr:  address of the qc values                            */
/* Globals:  info                                                          */
/* Returns:  OK if was able to get qc data                                 */
/*           ERROR if file doesn't exist or insufficient data              */
/* Calls:  OpenFile                                                        */
/*         ScanQC                                                          */
/*-------------------------------------------------------------------------*/
FILE *qc_fp;       /* pointer to QC file */
int open_result;   /* signifies if error occurred when opening file */
int ScanQC(FILE *, struct qc *);


void GetSumPrecTemp(FILE *yearlysum_fp, float *total_gen_precip,
                float *ave_gen_temp, int first_gen_day, int last_gen_day,
                int first_gen_day_year, int last_gen_day_year)
/*-------------------------------------------------------------------------*/
/* For a year of gen'd data, the ave. temp. and total precip. are          */
/*    calculated for the period over which gen'd data is required.         */
/* Receives: yearlysum_fp:  pointer to WGEN yearly summary file            */
/*           *total_gen_precip:  address of total precip                   */
/*           *ave_gen_temp:  address of ave. temp.                         */
/*           first_gen_day:  first day of gen'd data                       */
/*           last_gen_day:  last day of gen'd data                         */
/*           first_gen_day_year:  year of first day of gen'd data          */
/*           last_gen_day_year:  year of last day of gen'd data            */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: CalDate                                                          */
/*-------------------------------------------------------------------------*/
int last_gen_day_month; /* month of last day of gen'd data */
int first_gen_day_month; /* month of first day of gen'd data */
int day; /* day of month */
int divisor_for_averages = 0; /* number of months used to calculate ave. temp */
char line_from_file[81]; /* string used to read a line of data from file */
float yearlysum_precip; /* precip value read from WGEN yearly summary file */
float gen_maxt_total = 0.0; /* used to calculate ave. temp. */
float gen_mint_total = 0.0; /* used to calculate ave. temp. */
float yearlysum_maxt; /* max temp. value read from WGEN yearly summary file */
float yearlysum_mint; /* min temp. value read from WGEN yearly summary file */
int i; /* loop counter */


int GetValueToChange(struct formatted_day *formatted_ptr, int month, int day,
                int year)
/*-------------------------------------------------------------------------*/
/* Lets user choose which value of the daily data to edit.                 */
/* Receives: *formatted_ptr:  address of the formatted data                */
/*           month:  month number (1-12)                                   */
/*           day:  day of month                                            */
/*           year:  two-digit year                                         */
/* Globals: none.                                                          */
/* Returns: integer (1-4):  corresponding to value to view                 */
/*          integer value of q or Q:  indicating user wants to quit editing*/
/* Calls: ShowValueChoices                                                 */
/*-------------------------------------------------------------------------*/
int choice;   /* user's choice */
int ascii_number;      /* ASCII value of the user's choice */
void ShowValueChoices(struct formatted_day *, int, int, int);


char GetViewEditAction(void)
/*-------------------------------------------------------------------------*/
/* Gets user's decision to edit or view a file, or exit to the previous    */
/*  menu.                                                                  */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: v:  if user wants to view                                      */
/*          e:  if user wants to edit                                      */
/*          x:  if user wants to exit (return to previous menu)            */
```

```
/* Calls: none.                                                              */
/*---------------------------------------------------------------------------*/
int view_or_edit; /* indicates user's choice to view or edit a file */


void HourQC(struct hour *presenthour_ptr, struct hour *previous1hour_ptr,
            struct hour *previous2hour_ptr, struct qc *qc_ptr)
/*---------------------------------------------------------------------------*/
/* Performs QC procedures on hourly data sets.                               */
/* Receives: *presenthour_ptr:  address of the data of the present hour      */
/*                              being run through QC                         */
/*           *previous1hour_ptr:  address of the hourly summary data for     */
/*                                the hour previous to the present hour      */
/*           *previous2hour_ptr:  address of the hourly summary data for     */
/*                                two hours previous to the present hour     */
/*           *qc_ptr:  address of the qc values                              */
/* Globals: info                                                             */
/* Returns: none.                                                            */
/* Calls: CalDate                                                            */
/*        PrintWarning                                                       */
/*        HourMissingQC                                                      */
/*        HourRHQC                                                           */
/*        HourSRQC                                                           */
/*        HourTempQC                                                         */
/*---------------------------------------------------------------------------*/
int warning; /* number signifying the type of QC error found (if any) */
int month_for_showing; /* month of hourly value which failed QC */
int day_for_showing;  /* day of hourly value which failed QC */
int year_for_showing; /* year of hourly value which failed QC */
int HourMissingQC(int, int);
int HourTempQC(float, int, struct qc *, float, float);
int HourSRQC(float, int);
int HourRHQC(float, struct qc *, float, float);


int HourMissingQC(int presenthour_hour, int previous1hour_hour)
/*---------------------------------------------------------------------------*/
/* Determines if the present hour follows the previous hour by one hour.     */
/* Receives: presenthour_hour:  hour of the present hourly summary           */
/*           previous1hour_hour:  hour of the previous hourly summary        */
/* Globals: none.                                                            */
/* Returns: PASS if hour passes test                                         */
/*          HOURMISSING if hour fails test                                   */
/* Calls: none.                                                              */
/*---------------------------------------------------------------------------*/


int HourRHQC(float presenthour_rh, struct qc *qc_ptr, float previous1hour_rh,
             float previous2hour_rh)
/*---------------------------------------------------------------------------*/
/* Performs QC procedures on hourly rel. humidity values.                    */
/* Receives:  presenthour_rh: rel. hum. value of the present hourly summary*/
/*            *qc_ptr:  address of QC values                                 */
/*            previous1hour_rh:  rel. hum. value of previous hourly summary*/
/*            previous2hour_rh:  rel. hum. value of 2-hours previous         */
/* Returns: PASS if value passes test                                        */
/*          RHOUTRNG if value is out of range                                */
/*          RHCONSTNT if value is constant over time                         */
/* Calls: CheckConstant                                                      */
/*        CheckExtreme                                                       */
/*---------------------------------------------------------------------------*/


int HourSRQC(float presenthour_sr, int presenthour_hour)
/*---------------------------------------------------------------------------*/
/* Performs QC procedures on hourly ave. solar radiation value.              */
/* Receives: presenthour_sr:  solar rad. value of present hourly summary     */
/*           presenthour_hour:  hour of present hourly summary               */
/* Globals: none.                                                            */
/* Returns: PASS if value passes all tests                                   */
/*          SOLARPOSITIV if value is positive at night                       */
/* Calls: none.                                                              */
/*---------------------------------------------------------------------------*/
```

```
int HourTempQC(float presenthour_temp, int presenthour_month,
               struct qc *qc_ptr, float previous1hour_temp, float previous2hour_temp)
/*-------------------------------------------------------------------------*/
/* Performs QC procedures on hourly ave. temperature value.                */
/* Receives:  presenthour_temp:  ave. temp. value of present hourly summary*/
/*            presenthour_month:  month of present hourly summary          */
/*            *qc_ptr:  address of QC values                               */
/*            previous1hour_temp:  ave. temp. value of previous hourly     */
/*                                 summary                                 */
/*            previous2hour_temp:  ave. temp. value of 2-hours previous    */
/* Returns: PASS if value passes test                                      */
/*          AVETOUTRNG if value is out of range                            */
/*          AVETCONSTNT if value is constant over time                     */
/* Calls: CheckConstant                                                    */
/*        CheckExtreme                                                     */
/*-------------------------------------------------------------------------*/


void InitializeInfileValues(struct wgen_input *wgen_input_ptr)
/*-------------------------------------------------------------------------*/
/* Initializes WGEN input parameters for editing.                          */
/* Receives: *wgen_input_ptr:  address of data for WGEN input file         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int i; /* loop counter */


void InitializeProgramInfo(void)
/*-------------------------------------------------------------------------*/
/* Initializes program information.                                        */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


int JulDate(int month, int day, int year)
/*-------------------------------------------------------------------------*/
/* Converts month, day and year to a julian date.                          */
/* Receives:  month:  number of month (1-12)                               */
/*            day:  day of month                                           */
/*            year:  two-digit year                                        */
/* Globals:  none.                                                         */
/* Returns:  the julian date, if successful                                */
/*           JD_INVALID_MONTH if invalid month                             */
/*           JD_INVALID_DAY if invalid day                                 */
/*           JD_INVALID_YEAR if invalid year                               */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
int i;  /* loop counter */
int calculated_jdate = 0;  /* Julian date calculated from calendar date */
static int days_per_month[12] = { 31, 28, 31, 30, 31, 30,


int LoadInfileData(struct wgen_input *wgen_input_ptr)
/*-------------------------------------------------------------------------*/
/* Loads WGEN input parameters for editing.                                */
/* Receives: *wgen_input_ptr:  address of data for WGEN input file         */
/* Globals: none.                                                          */
/* Returns: OK:  if info successfully loaded                               */
/*          ERROR:  if some error occured                                  */
/* Calls: ScanInfileData                                                   */
/*-------------------------------------------------------------------------*/
FILE *wgeninput_fp;  /* pointer for WGEN input file */
int open_result;   /* signifies if error occured when opening file */
int ScanInfileData(FILE *, struct wgen_input *);


int LoadProgramInfo(void)
```

```
/*-------------------------------------------------------------------------*/
/* If program info file exists, info is retrieved from file.               */
/* Receives: none.                                                         */
/* Globals:  none.                                                         */
/* Returns: OK if info is loaded                                           */
/*          ERROR if info file does not exist                              */
/* Calls: ScanProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
FILE *info_fp;     /* pointer to program info file */
int open_result;   /* signifies if error occured during file opening */
int ScanProgramInfo(FILE *);


main()
/*-------------------------------------------------------------------------*/
/* Prints main menu, gets the user's choice of what to do, and calls the   */
/*  appropriate function.  Loads program information.                      */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: value of 1 if program exits properly                           */
/* Calls: ShowMainMenu                                                     */
/*        RunQC                                                            */
/*        GetGeneratedData                                                 */
/*        FileViewEdit                                                     */
/*        MiscFunctions                                                    */
/*        LoadProgramInfo                                                  */
/*        EditProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
int choice;  /* user's choice */
int LoadProgramInfo(void);
void ShowMainMenu(void);
void RunQC(void);
void FileViewEdit(void);
void GetGeneratedData(void);
void MiscFunctions(void);


void MiscFunctions()
/*-------------------------------------------------------------------------*/
/* Lets user monitor the weather station and set the datalogger's clock;   */
/*  edit some of the program information; or start a new year of data      */
/*  collection.                                                            */
/* Receives: none.                                                         */
/* Globals: none                                                           */
/* Returns: none.                                                          */
/* Calls: ShowMiscMenu                                                     */
/*        MonitorStation                                                   */
/*        EditProgramInfo                                                  */
/*        StartNewDataYear                                                 */
/*-------------------------------------------------------------------------*/
int choice; /* user's choice */
void ShowMiscMenu(void);
void MonitorStation(void);
void StartNewDataYear(void);


void MonitorStation(void)
/*-------------------------------------------------------------------------*/
/* Allows user to monitor weather station in operation, and to set the     */
/*  datalogger clock.                                                      */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int choice; /* user's choice */


void MoveToChosenYear(FILE *dailysum_fp, int gen_year_number)
/*-------------------------------------------------------------------------*/
/* Moves a file pointer to the beginning of the chosen year of gen'd data  */
/*  in the WGEN daily summary file.                                        */
/* Receives: *dailysum_fp:  file pointer to the WGEN daily summary file    */
```

```
/*          gen_year_number:  number of the year of gen'd data chosen     */
/*                            by user                                      */
/* Globals: none.                                                         */
/* Returns: none.                                                         */
/* Calls: GetGenDay                                                       */
/*------------------------------------------------------------------------*/
int gen_year_end; /* number of days in the year of gen'd data chosen by user */
struct gen_data gendata; /* gen'd data for a day */


FILE *OpenCurrentFile(void)
/*------------------------------------------------------------------------*/
/* Opens file of current weather data                                     */
/* Receives: none.                                                        */
/* Globals: info                                                          */
/* Returns: file pointer to the opened file                               */
/* Calls: none.                                                           */
/*------------------------------------------------------------------------*/
FILE *current_fp;  /* pointer to the current data file */


FILE *OpenFile(int *open_result, char *file_name_to_open,
               char *file_open_activity)
/*------------------------------------------------------------------------*/
/* Opens a file for a given activity.                                     */
/* Receives:  *open_result:  address of flag if error occurred during file */
/*                           opening                                      */
/*            *file_name_to_open:  address of name of file to open        */
/*            *file_open_activity:  address of how to open a file (read,   */
/*                                  write, append                         */
/* Globals:  none.                                                        */
/* Returns:  file pointer to the opened file                              */
/* Calls:  none.                                                          */
/*------------------------------------------------------------------------*/
FILE *fp;  /* pointer to file */


FILE *OpenPreviousFile(struct day *previousday_ptr,
               struct hour *previous1hour_ptr, struct hour *previous2hour_ptr)
/*------------------------------------------------------------------------*/
/* Opens file with previous data.                                         */
/* Receives: *previousday_ptr:  address of the collected daily summary    */
/*                              data for the day previous to the present  */
/*                              day being run through QC                  */
/*           *previous1hour_ptr:  address of the hourly summary data for  */
/*                                the hour previous to the present hour   */
/*           *previous2hour_ptr:  address of the hourly summary data for  */
/*                                two hours previous to the present hour  */
/* Globals: info                                                          */
/* Returns: file pointer to the opened file                               */
/* Calls: ScanDay                                                         */
/*        ScanHour                                                        */
/*------------------------------------------------------------------------*/
FILE *previous_fp; /* pointer to the previous data file */


void PrintArrayID(FILE *backup_fp, int arrayid)
/*------------------------------------------------------------------------*/
/* Prints the array ID to the back-up file.                               */
/* Receives: *backup_fp: pointer to the back-up file                      */
/*           arrayid:  ID number for hourly, daily, or precip occurrance  */
/*                     summary                                            */
/* Globals: none.                                                         */
/* Returns: none.                                                         */
/* Calls: none.                                                           */
/*------------------------------------------------------------------------*/


void PrintDay(FILE *fp, struct day *daydata_ptr)
/*------------------------------------------------------------------------*/
/* Prints daily data to backup file.                                      */
/* Receives: *fp: pointer to file                                         */
/*           *daydata_ptr:  address of the data of a daily summary        */
```

```
/* Globals: none.                                                          */
/* Returns: none.                                                         */
/* Calls: none.                                                           */
/*------------------------------------------------------------------------*/


void PrintDaysDataToScreen(struct day *daydata_ptr)
/*------------------------------------------------------------------------*/
/* Prints the IBSNAT values of a day's data set to the screen.            */
/* Receives: *daydata_ptr:  address of data of a daily summary            */
/* Globals: value_code                                                    */
/*          units_code                                                    */
/* Returns: none.                                                         */
/* Calls: none.                                                           */
/*------------------------------------------------------------------------*/


void PrintFormattedDay(FILE *fp, struct formatted_day *formatted_ptr,
               int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION])
/*------------------------------------------------------------------------*/
/* Prints IBSNAT data, the calendar date the data is for, and any error/  */
/*  action codes associated with the data to an IBSNAT formatted file.    */
/* Receives:  *fp:  pointer to file                                       */
/*            *formatted_ptr:  address of formatted daily data            */
/*            erroractions[][]:  array of qc errors and actions           */
/* Globals:  month_name                                                   */
/* Returns:  none.                                                        */
/* Calls:  CalDate                                                        */
/*------------------------------------------------------------------------*/
int i;   /* loop counter */
int j;   /* loop counter */
int month;    /* month number (1-12) */
int day;      /* day of month */


void PrintHour(FILE *fp, struct hour *hourdata_ptr)
/*------------------------------------------------------------------------*/
/* Prints hourly data to the back-up file.                                */
/* Receives: *fp: pointer to file                                         */
/*           *hourdata_ptr:  address of data of an hourly summary         */
/* Globals: none.                                                         */
/* Returns: none.                                                         */
/* Calls: none.                                                           */
/*------------------------------------------------------------------------*/


void PrintJulianDateWarning(int warning, struct day *presentday_ptr,
               int presentday_month, int presentday_day, int previousday_month,
               int previousday_day, int previousday_year)
/*------------------------------------------------------------------------*/
/* Prints a warning for a Julian date QC error to the screen.             */
/* Receives: warning:  number signifying the type of QC error (if any)    */
/*           *presentday_ptr:  address of the data of the present day     */
/*                             being run through QC                       */
/*           presentday_month:  month of the present daily summary        */
/*           presentday_day:  day of the present daily summary            */
/*           previousday_month:  month of the previous daily summary      */
/*           previousday_day:  day of the present daily summary           */
/*           previousday_year:  year of the previous daily summary        */
/* Globals: error_code                                                    */
/*          month_name                                                    */
/* Returns: none.                                                         */
/* Calls: PrintDaysDataToScreen                                           */
/*------------------------------------------------------------------------*/


void PrintPrecip(FILE *fp, int precip_time, float precip_amount)
/*------------------------------------------------------------------------*/
/* Prints precip data to the back-up file.                                */
/* Receives: fp: file pointer to the back-up file                         */
/*           precip_time: time of the precip occurrance                   */
/*           precip_amount: amount of the precip occurrance               */
/* Globals: none.                                                         */
```

```
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


void PrintProgramInfo(void)
/*-------------------------------------------------------------------------*/
/* Displays program information on screen.                                 */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


void PrintWarning(int warning, int month, int day, int year, int hour,
                  float warning_value, int val_code, int unit_code)
/*-------------------------------------------------------------------------*/
/* Prints a warning for a QC error to the screen.                          */
/* Receives: warning:  code signifying the QC error                        */
/*           month:  month number (1-12)                                   */
/*           day:  day of month                                            */
/*           year:  two-digit year                                         */
/*           hour:  hour in 24-hour format                                 */
/*           warning_value:  value which caused the QC error               */
/*           val_code:  number signifying what the value represents        */
/*           unit_code:  code signifying units for a value                 */
/* Globals: value_code                                                     */
/*          error_code                                                     */
/*          units_code                                                     */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


int QCNewValue(float checkvalue, float previous1value, int val_code,
               int checkvalue_month, struct qc *qcptr, float opposite_value)
/*-------------------------------------------------------------------------*/
/* Checks a new value entered by user to see if it is reasonable.          */
/* Receives:  checkvalue:  the value to check                              */
/*            previous1value:  immediately previous value for comparison   */
/*            val_code:  number signifying what the value represents       */
/*            checkvalue_month:  the month the value was collected         */
/*            *qcptr:  address of the qc values                            */
/*            opposite_value:  the opposite value for comparison           */
/* Globals:  error_code[]                                                  */
/* Returns:  PASS if the value passes all qc tests or if value fails a     */
/*           test and user wants to keep the value                         */
/*           FAIL if the value fails any qc test and user does not want to */
/*           keep the value                                                */
/* Calls:  DayTempQC                                                       */
/*         DaySRQC                                                         */
/*         DayPrecQC                                                       */
/*-------------------------------------------------------------------------*/
int warning;  /* number indicating type of QC error found (if any) */
int choice;   /* user's choice */


void RunGen(void)
/*-------------------------------------------------------------------------*/
/* Runs WGEN to produce a daily summary file and a yearly summary file.    */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int rungen_result; /* signifies if an error occurred while running WGEN */


void RunQC(void)
/*-------------------------------------------------------------------------*/
/* Gets any unretrieved weather data from station (stored in ASCII file),  */
/*  reads one data set at a time, stores each in backup file, runs qc      */
```

```
/*  procedures on hourly and daily data sets, stores daily summaries in    */
/*  IBSNAT format in the current file, then deletes the raw data file.     */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns:  none.                                                         */
/* Calls:  CallStation                                                     */
/*         FormatData                                                      */
/*         ConvertRawData                                                  */
/*         DayQC                                                           */
/*         GetDay                                                          */
/*         GetHour                                                         */
/*         GetPrecip                                                       */
/*         GetQCValues                                                     */
/*         HourQC                                                          */
/*         OpenFile                                                        */
/*         OpenCurrentFile                                                 */
/*         OpenPreviousFile                                                */
/*         PrintArrayID                                                    */
/*         PrintDay                                                        */
/*         PrintFormattedDay                                               */
/*         PrintHour                                                       */
/*         PrintPrecip                                                     */
/*         SavePrevious                                                    */
/*         ScanArrayID                                                     */
/*-------------------------------------------------------------------------*/
struct day day_data[DAYS_PREVIOUS];  /* present and previous daily summaries */
struct hour hour_data[HOURS_PREVIOUS]; /* present, previous hourly summaries */
struct formatted_day formatted_data; /* daily summary in IBSNAT format */
struct qc qcvalues;  /* values for QC tests */
int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION];

                        /* array of QC errors and actions */
int precip_time;  /* time of a precipitation occurrance */
float precip_amount;  /* amount of a precipitation occurrance */
int arrayid;    /* ID for hourly, daily, or precip occurrance summary */
FILE *rawdata_fp;     /* pointer to raw data file */
FILE *previous_fp;    /* pointer to previous data file */
FILE *backup_fp;      /* pointer to backup data file */
FILE *current_fp;     /* pointer to file of current IBSNAT-formatted data */
int open_result;      /* signifies if error occured during file opening */
int choice;           /* user's choice */
void CallStation(void);
void ConvertRawData(struct day *);
int DayQC(struct day *, struct day *, struct qc *,
              int [VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION], FILE *);
void GetDay(FILE *, struct day *);
void GetHour(FILE *, struct hour *);
void GetPrecip(FILE *, int *, float *);
void HourQC(struct hour *, struct hour *, struct hour *, struct qc *);
FILE *OpenCurrentFile(void);
FILE *OpenPreviousFile(struct day *, struct hour *, struct hour *);
void PrintArrayID(FILE *, int);
void PrintPrecip(FILE *, int, float);
void SavePrevious(FILE *, struct day *, struct hour *, struct hour *);
int ScanArrayID(FILE *);


void SaveInfileData(struct wgen_input *wgen_input_ptr)
/*-------------------------------------------------------------------------*/
/* Saves WGEN input parameters in a file.                                  */
/* Receives: *wgen_input_ptr:  address of data for WGEN input file         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: OpenFile                                                         */
/*-------------------------------------------------------------------------*/
int i;   /* loop counter  */
FILE *wgeninput_fp; /* pointer to WGEN input file */
int open_result; /* signifies if error occurred during file opening */
int print_result; /* signifies if error occurred when printing to file */


void SavePrevious(FILE *previous_fp, struct day *previousday_ptr,
              struct hour *previous1hour_ptr, struct hour *previous2hour_ptr)
```

```
/*-------------------------------------------------------------------------*/
/* Saves data used for future qc in previous file.                         */
/* Receives: previous_fp: pointer to the previous data file                */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           *previous1hour_ptr:  address of the hourly summary data for   */
/*                                the hour previous to the present hour    */
/*           *previous2hour_ptr:  address of the hourly summary data for   */
/*                                two hours previous to the present hour   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: PrintDay                                                         */
/*        PrintHour                                                        */
/*-------------------------------------------------------------------------*/


void SaveProgramInfo(void)
/*-------------------------------------------------------------------------*/
/* Saves program information to a file.                                    */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
FILE *info_fp;  /* pointer to program info file */
int open_result;  /* indicates if error occurred during file opening */
int print_result;  /* indicates if error occurred while printing to file */


void SaveQCValues(struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Saves QC values in a file.                                              */
/* Receives: *qcptr:  address of qc values                                 */
/* Globals: info                                                           */
/* Returns:  none.                                                         */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
int i; /* loop counter */
int j = 0; /* loop counter */
FILE *qc_fp; /* pointer to QC values file */
int open_result; /* signifies if error occurred during file opening */


int ScanArrayID(FILE *raw_fp)
/*-------------------------------------------------------------------------*/
/* Reads the array ID from the raw data file.                              */
/* Receives: *raw_fp: pointer to the raw data file                         */
/* Globals: none.                                                          */
/* Returns: the ID number read                                             */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int arrayid;


int ScanDay(FILE *fp, struct day *daydata_ptr)
/*-------------------------------------------------------------------------*/
/* Reads in line of daily data from the raw data file or previous file.    */
/* Receives: *fp: pointer to file                                          */
/*           *daydata_ptr:  address of data of a daily summary             */
/* Globals: none.                                                          */
/* Returns: OK if data was read                                            */
/*          FILE_END if error occurred                                     */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


int ScanHour(FILE *fp, struct hour *hourdata_ptr)
/*-------------------------------------------------------------------------*/
/* Reads in line of hourly data from raw data file or previous file.       */
/* Receives: *fp: pointer to file                                          */
/*           *hourdata_ptr:  address of data of an hourly summary          */
/* Globals: none.                                                          */
```

```
/* Returns: OK if data was read                                          */
/*          FILE_END if error occurred                                   */
/* Calls: none.                                                          */
/*-----------------------------------------------------------------------*/


int ScanInfileData(FILE *wgeninput_fp, struct wgen_input *wgen_input_ptr)
/*-----------------------------------------------------------------------*/
/* Gets data from WGEN input file and puts it into wgen_input struct.    */
/* Receives:  *wgeninput_fp:  pointer to WGEN input file                 */
/*            *wgen_input_ptr:  address of data for WGEN input file      */
/* Returns:  OK:  if data successfully loaded                            */
/*           FILE_END:  if inadequate data in file                       */
/* Calls:  none                                                          */
/*-----------------------------------------------------------------------*/
int i;   /* loop counter */


int ScanProgramInfo(FILE *info_fp)
/*-----------------------------------------------------------------------*/
/* Gets data from program info file and puts it into global struct.      */
/* Receives:  *info_fp:  pointer to program info file                    */
/* Globals: info                                                         */
/* Returns:  OK if data successfully loaded                              */
/*           FILE_END if inadequate data in file                         */
/* Calls:  none.                                                         */
/*-----------------------------------------------------------------------*/
char dummy_string[2]; /* used to read carriage return at end of line */


int ScanQC(FILE *qc_fp, struct qc *qcptr)
/*-----------------------------------------------------------------------*/
/* Gets data from QC file and puts it into QC struct.                    */
/* Receives:  *qc_fp:  pointer to qc file                                */
/*            *qcptr:  address of qc data                                */
/* Globals: none.                                                        */
/* Returns:  OK if qc data successfully loaded                           */
/*           FILE_END if inadequate data in file                         */
/* Calls:  none.                                                         */
/*-----------------------------------------------------------------------*/
int i;   /* loop counter */
int j = 0;   /* loop counter */
char line_from_file[100]; /* string to read a line of data from file */


void SetGeneratedStartingDay(int last_current_day, int last_current_day_year,
                int *first_gen_day, int *first_gen_day_year)
/*-----------------------------------------------------------------------*/
/* Figures the starting day for generated data as the day after current  */
/*    data ends.                                                         */
/* Receives: last_current_day:  last day of current data                 */
/*           last_current_day_year:  year of last day of current data    */
/*           *first_gen_day:  value of first day of gen'd data to be placed*/
/*              in this address when calculated                          */
/*           *first_gen_day_year:  value of year of first day of gen'd data*/
/*              to be placed in this address when calculated             */
/* Globals: none.                                                        */
/* Returns: none.                                                        */
/* Calls: none.                                                          */
/*-----------------------------------------------------------------------*/


void ShowFile(char *file_name_to_view)
/*-----------------------------------------------------------------------*/
/* Puts file contents on screen or printer, line by line.                */
/* Receives: *file_name_to_view:  address of name of file to view        */
/* Globals: none.                                                        */
/* Returns: none.                                                        */
/* Calls: OpenFile                                                       */
/*        GetDataLineFromFile                                            */
/*-----------------------------------------------------------------------*/
int choice; /* user's choice */
FILE *view_in_fp;  /* Pointer to data file to view */
```

```
FILE *view_out_fp; /* Pointer to output file for viewing (screen or printer) */
int screen_view = TRUE; /* signifies if user is viewing on screen or printer */
int open_result; /* indicates if error occurred during file opening */
char line_from_file[100];   /* string to read a line of data from file */
int line_count = 0; /* used to determine when screen is full of data */
int GetDataLineFromFile(FILE *, char [100]);


void ShowMainMenu(void)
/*-------------------------------------------------------------------------*/
/* Prints main menu to screen.                                             */
/* Receives: none.                                                         */
/* Globals:  none.                                                         */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


void ShowMiscMenu()
/*-------------------------------------------------------------------------*/
/* Prints misc. functions menu to screen.                                  */
/* Receives: none.                                                         */
/* Globals:  none                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


void ShowValueChoices(struct formatted_day *formatted_ptr, int month, int day,
                int year)
/*-------------------------------------------------------------------------*/
/* Displays values for a day of data with corresponding numbers so user    */
/*  may choose which value to edit.                                        */
/* Receives: *formatted_ptr:  address of the formatted data                */
/*           month:  month number (1-12)                                   */
/*           day:  day of month                                            */
/*           year:  two-digit year                                         */
/* Globals: value_code                                                     */
/*          units_code                                                     */
/*          month_name                                                     */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/


void Sort(int number_to_sort, float first_array[], float second_array[],
                int third_array[])
/*-------------------------------------------------------------------------*/
/* Insertion sort function; sorts on key array and makes the same changes  */
/*  to the two other arrays to keep corresponding data together.           */
/* Receives: number_to_sort:  number of values to sort                     */
/*           first_array:  array of precip or temp summaries to sort by    */
/*           second_array:  follows sorting of first array                 */
/*           third_array:  array of year numbers; follows first array      */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int i; /* loop counter */
int j; /* loop counter */
float temporary1; /* holds value being sorted */
float temporary2; /* holds value being sorted */
int temporary3; /* holds value being sorted */


void StartNewDataYear(void)
/*-------------------------------------------------------------------------*/
/* Allows user to begin a new year of data collection.                     */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: EditProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
```

```
int choice; /* user's choice */


void ZeroQCValues(struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Sets all QC values to zero.                                             */
/* Receives: *qcptr:  address of qc values                                 */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
int i; /* loop counter */
int j;  /* loop counter */
```

PROGRAMMER'S GUIDE

Function Calls/Called By List

Function	Calls	Called By
AddDayToMissingDate	SaveProgramInfo	FillMissingData
AllGeneratedDataStartDay	JulDate	GetGeneratedData
AskToUseAllGeneratedData	none	GetGeneratedData
AveragePresentAndPreviousData	none	FillMissingData
CalDate	none	ChooseHowToFillMissingData DayQC DayJulianDateQC DecideOnFirstDaysData HourQC GetSumPrecTemp PrintFormattedDay
CallStation	none	RunQC
ChangeDateOrDeleteData	none	DayJulianDateQC
ChangeDateOrFillMissingData	PrintDaysDataToScreen	DayJulianDateQC
CheckChange	none	DayTempQC
CheckConstant	none	DayTempQC DayRHQC DayWindQC HourRHQC HourTempQC
CheckExtreme	none	DayMoisQC DayPrecQC DayRHQC DaySRQC DayTempQC DayVoltQC HourRHQC HourTempQC
CheckMaxMin	none	DayRHQC DayTempQC
ChooseHowToFillMissingData	CalDate PrintDaysDataToScreen	FillMissingData
ConvertForQC	none	GetNewValue
ConvertRawData	none	RunQC
DayJulianDateQC	CalDate DecideOnFirstDaysData PrintJulianDateWarning ChangeDateOrDeleteData GetNewDateFromUser GetDifference ChangeDateOrFillMissingData	DayQC
DayMoisQC	CheckExtreme	DayQC
DayQC	FigureProperYear DayJulianDateQC CalDate	RunQC

	DayTempQC PrintWArning FixError DaySRQC DayPrecQC DayRHQC DayWindQC DayVoltQC DayMoisQC FillMissingData	
DayRHQC	CheckConstant CheckExtreme CheckMaxMin	DayQC
DayPrecQC	CheckExtreme	DayQC QCNewValue
DaySRQC	CheckExtreme	DayQC QCNewValue
DayTempQC	CheckExtreme CheckConstant CheckChange CheckMaxMin	DayQC QCNewValue
DayVoltQC	CheckExtreme	DayQC
DayWindQC	CheckConstant	DayQC
DecideOnFirstDaysData	CalDate PrintDaysDataToScreen GetNewFirstDateFromUser SaveProgramInfo	DayJulianDateQC
DisplaySumData	GetSumPrecTemp Sort	GetGeneratedData
EditCurrentDataFile	JulDate EditDayOfCurrentData	FileViewEdit
EditDayOfCurrentData	OpenFile GetFormattedDay PrintFormattedDay GetQCVAlues GetValueToChange GetNewValue	EditCurrentDataFile
EditProgramInfo	InitializeProgramInfo OpenFile PrintProgramInfo SaveProgramInfo	main StartNewDataYear MiscFunctions
EditQCValues	GetQCValues ZeroQCValues EnterQCMaxMin SaveQCValues	FileViewEdit
EnterQCMaxMin	none	EditQCValues
FigureProperYear	SaveProgramInfo	DayQC
FileViewEdit	FileViewEditMenu GetViewEditAction ShowFile EditCurrentDataFile EditQCValues	main
FileViewEditMenu	none	FileViewEdit
FillMissingData	ChooseHowToFillMissingData AveragePresentAndPreviousData	DayQC

	AddDayToMissingDate FormatData PrintFormattedDay	
FixError	QCNewValue	DayQC
FormatData	none	RunQC FillMissingData
FormatGenData	none	GetGeneratedData
GetDay	ScanDay	RunQC
GetDataLineFromFile	none	ShowFile
GetDifference	none	DayJulianDateQC
GetFormattedDay	none	EditDayOfCurrentData GetGeneratedData
GetGenDay	none	GetGeneratedData MoveToChosenYear
GetGeneratedData	GetNewInfileData OpenFile AskToUseAllGeneratedData GetFormattedDay PrintFormattedDay SetGeneratedStartingDay AllGeneratedDataStartDay RunGen DisplaySumData MoveToChosenYear FormatGenData GetGenDay	main
GetHour	ScanHour	RunQC
GetInfileValue	none	GetNewInfileData
GetNewDateFromUser	JulDate SaveProgramInfo	DayJulianDateQC
GetNewFirstDateFromUser	JulDate	DecideOnFirstDaysData
GetNewInfileData	LoadInfileData InitializeInfileData GetInfileValue SaveInfileData	GetGeneratedData
GetNewValue	ConvertForQC QCNewValue	EditDayOfCurrentData
GetPrecip	none	RunQC
GetQCValues	OpenFile ScanQC	EditDayOfCurrentData EditQCValues RunQC
GetSumPrecTemp	CalDate	DisplaySumData
GetValueToChange	ShowValueChoices	EditDayOfCurrentData
GetViewEditAction	none	FileViewEdit
HourQC	CalDate PrintWarning HourMissingQC HourTempQC HourSRQC HourRHQC	RunQC
HourMissingQC	none	HourQC

HourRHQC	CheckConstant CheckExtreme	HourQC
HourSRQC	none	HourQC
HourTempQC	CheckConstant CheckExtreme	HourQC
InitializeInfileValues	none	GetNewInfileData
InitializeProgramInfo	none	EditProgramInfo
JulDate	none	EditCurrentDataFile AllGeneratedDataStartDay GetNewFirstDateFromUser GetNewDateFromUser
LoadInfileData	ScanInfileData	GetNewInfileData
LoadProgramInfo	ScanProgramInfo	main
main	LoadProgramInfo EditProgramInfo ShowMainMenu RunQC FileViewEdit GetGeneratedData MiscFunctions	none
MiscFunctions	ShowMiscMenu MonitorStation EditProgramInfo StartNewDataYear	main
MonitorStation	none	MiscFunctions
MoveToChosenYear	GetGenDay	GetGeneratedData
OpenCurrentFile	none	RunQC
OpenFile	none	GetQCValues EditDayOfCurrentData GetGeneratedData ShowFile EditProgramInfo RunQC
OpenPreviousFile	ScanHour ScanDay	RunQC
PrintArrayID	none	RunQC
PrintDay	none	RunQC SavePrevious
PrintDaysDataToScreen	none	PrintJulianDateWarning DecideOnFirstDaysDate ChangeDateOrFillMissingData ChooseHowToFillMissingData
PrintFormattedDay	CalDate	EditDayOfCurrentData GetGeneratedData RunQC FillMissingData
PrintHour	none	RunQC SavePrevious
PrintJulianDateWarning	PrintDaysDataToScreen	DayJulianDateQC
PrintPrecip	none	RunQC
PrintProgramInfo	none	EditProgramInfo

PrintWarning	none	RunQC HourQC
QCNewValue	DayTempQC DaySRQC DayPrecQC	FixError GetNewValue
RunGen	none	GetGeneratedData
RunQC	CallStation ConvertRawData OpenFile GetQCValues OpenPreviousFile ScanArrayID PrintArrayID GetHour PrintHour HourQC GetDay PrintDay DayQC FormatData PrintFormattedDay GetPrecip PrintPrecip SavePrevious	main
SaveInfileData	none	GetNewInfileData
SavePrevious	PrintHour PrintDay	RunQC
SaveProgramInfo	none	EditProgramInfo AddDayToMissingDate DecideOnFirstDaysData FigureProperYear GetNewDateFromUser
SaveQCValues	none	EditQCValues
ScanArrayID	none	RunQC
ScanDay	none	OpenPreviousFile GetDay
ScanHour	none	OpenPreviousFile GetHour
ScanInfileData	none	LoadInfileData
ScanProgramInfo	none	LoadProgramInfo
ScanQC	none	GetQCValues
SetGeneratedStartDay	none	GetGeneratedData
ShowFile	OpenFile GetDataLineFromFile	FileViewEdit
ShowMainMenu	none	main
ShowMiscMenu	none	MiscFunctions
ShowValueChoices	none	GetValueToChange
Sort	none	DisplaySumData
StartNewDataYear	EditProgramInfo	MiscFunctions
ZeroQCValues	none	EditQCValues

PROGRAMMER'S GUIDE

Descriptions for Data Manager Program Files

The program information file.

Format:

```
sta_name\n
raw_file\n
prev_file\n
backup_file\n
current_file\n
curgen_file\n
qc_file\n
location_id\n
station_id\n
"%6.2f %6.2f\n", lat,lon
"%2d\n", year
```

Example:

```
sta
sta.dat
sta90.pr
sta90.bu
sta90.cu
osst0912.w90
sta90.qc
os
st
 44.35 123.10
90
```

The raw data file (created by the CSI TELCOM program) and the back-up file are exactly the same format, except that in the back-up file, the data are set to either two or three decimal places.

Format (precip. occurrances):

```
"%d,%d,%.3f\n", arrayid,time,amount
```

Format (hourly summaries):

```
"%d,%d,%d,%.2f,%.2f,%.3f,%.2f,%.2f,%.2f\n", arrayid,jdate,hour,
temp,rh,sr,ws,wd,wstd
```

Format (daily summaries):

```
"%d,%d,%d,%.2f,%.2f,%.2f,%.2f,%.2f,%.3f,%.2f,%.2f,%.2f,%.2f,
%.2f,%.2f,%.2f\n", arrayid,jdate,hour,avet,maxt,mint,maxrh,
minrh,sr,maxws,avews,prec,mois,volt,crmaxt,crmint
```

Example (raw data):

```
124,1958,.01
129,344,2000,46.2,90.8,0,4.894,285.9,8.77
129,344,2100,45.3,92.2,0,1.175,245.3,20.22
129,344,2200,43.81,93.9,0,.675,234.8,52.05
129,344,2300,42.91,94.2,0,2.991,247.4,22.32
129,345,0,41.43,94.6,0,3.101,245.5,23.37
139,345,0,48.85,50.92,20.26,96.5,78,.034,17.34,4.025,.31,0,
12.29,10.45,3.367
```

Example (back-up data):

```
124,1958,0.010
129,344,2000,46.20,90.80,0.000,4.89,285.90,8.77
129,344,2100,45.30,92.20,0.000,1.17,245.30,20.22
129,344,2200,43.81,93.90,0.000,0.68,234.80,52.05
129,344,2300,42.91,94.20,0.000,2.99,247.40,22.32
129,345,0,41.43,94.60,0.000,3.10,245.50,23.37
139,345,0,48.85,50.92,20.26,96.50,78.00,0.034,17.34,4.03,
0.31,0.00,12.29,10.45,3.37
```

The QC values file.

Format:

```
   (temperature; print identifier after first months values)
   "%6.2f,%6.2f     (temperature)\n", temp[i][j],temp[i][j+1]
   (then print each months values on separate line)
   "%6.2f,%6.2f\n", temp[i][j],temp[i][j+1]

   (solar rad; print identifier after first months values)
   "%6.2f,%6.2f     (solar-radiation)\n", sr[i][j],sr[i][j+1]
   (then print each months values on separate line)
   "%6.2f,%6.2f\n", sr[i][j],sr[i][j+1]

   (precip; print identifier after first months values)
   "%6.2f,%6.2f     (precipitation)\n", prec[i][j],prec[i][j+1]
   (then print each months values on separate line)
   "%6.2f,%6.2f\n", prec[i][j],prec[i][j+1]

   "%6.2f,%6.2f     (relative-humidity)\n", rh[j],rh[j+1]
   "%6.2f,%6.2f     (datalogger-voltage)\n", volt[j],volt[j+1]
   "%6.2f,%6.2f     (datalogger-moisture)\n", mois[j],mois[j+1]
   "%6.2f          (temperature-change)\n", maxtempchng
```

Example:

```
    64.00, -1.00     (temperature)
    69.00, -5.00
    82.00, 12.00
    91.00, 24.00
    96.00, 28.00
   102.00, 32.00
   107.00, 36.00
   108.00, 37.00
   103.00, 27.00
    92.00, 22.00
    73.00, 10.00
    66.00,-14.00
     8.50,  0.50     (solar-radiation)
    13.80,  0.60
    22.60,  1.50
    24.60,  1.00
    30.60,  3.50
    30.10,  4.50
    29.60,  7.00
    25.10,  4.20
    22.60,  3.00
    15.10,  1.50
    10.00,  0.50
     6.50,  0.20
     4.30,  0.00     (precipitation)
     2.80,  0.00
     1.90,  0.00
     2.10,  0.00
     1.60,  0.00
     2.10,  0.00
     1.80,  0.00
     1.50,  0.00
     2.20,  0.00
     4.30,  0.00
     3.20,  0.00
     5.00,  0.00
   100.00, 10.00     (relative-humidity)
    14.50, 11.00     (datalogger-voltage)
   200.00,  0.00     (datalogger-moisture)
    20.00           (temperature-change)
```

The previous data file:

```
Format:
   (for one hour previous)
   "%d,%d,%d,%.2f,%.2f,%.3f,%.2f,%.2f,%.2f\n", arrayid,jdate,hour,
      temp,rh,sr,ws,wd,wstd
   (for two hours previous)
   "%d,%d,%d,%.2f,%.2f,%.3f,%.2f,%.2f,%.2f\n", arrayid,jdate,hour,
      temp,rh,sr,ws,wd,wstd
   (for one day previous)
   "%d,%d,%d,%.2f,%.2f,%.2f,%.2f,%.2f,%.3f,%.2f,%.2f,%.2f,%.2f,
      %.2f,%.2f,%.2f\n", arrayid,jdate,hour,avet,maxt,mint,maxrh,
      minrh,sr,maxws,avews,prec,mois,volt,crmaxt,crmint

Example:
   345,1200,42.43,94.50,0.138,4.07,215.10,21.65
   345,1100,40.52,95.60,0.206,3.52,229.80,27.01
   344,0,48.85,50.92,20.26,96.50,78.00,2.938,17.34,4.03,
   0.31,0.00,12.29,10.45,3.37
```

The current and current/generated data files are exactly the same format except that generated data have a 'G' following the calendar date and also no error/action codes.

```
Format:
   "%2s%2s %2d %3d %5.2f %5.1f %5.1f %5.1f %6.2f", location_id,
   station_id,fyear,fjdate,fsr,fmaxt,fmint,fprec,pardat
   "  %s %2d", month_name[month],day
   (if data is generated, signify)
   "  G"
   (else if there are any error/action codes, print them)
   "  %d,%d", erroractions[i][QC_ERROR],erroractions[i][QC_ACTION]
   (print carriage return)
   "\n"

Example (current data):
   osst  44.35 123.10 12.07  0.00
   osst 90 341  2.07   4.2   2.2   0.0   0.00  Dec  7
   osst 90 342  7.78  23.6   3.2   0.0   0.00  Dec  8  6,1  15,1
   osst 90 343  0.60  10.5   8.1   2.2   0.00  Dec  9  8,1
Example (current/generated data):
   osst  44.35 123.10 12.07  0.00
   osst 90 341  2.07   4.2   2.2   0.0   0.00  Dec  7
   osst 90 342  7.78  23.6   3.2   0.0   0.00  Dec  8  6,1  15,1
   osst 90 343  0.60  10.5   8.1   2.2   0.00  Dec  9  8,1
   osst 90 344  1.76   6.1  -5.6   0.4   0.00  Dec 10  G
   osst 90 345  1.76   7.8   0.6   0.4   0.00  Dec 11  G
```

The WGEN input parameters file.

Format:

```
user_comments\n
"%5d%5d%5.1f%5d%5d\n",n_years, gen_code, lat_deg, tcf_code,
   rcf_code
(monthly values for probability of wet day given wet day)
"%6.3f", probwet_wet[i]
"\n"
(monthly values for probability of wet day given dry day)
"%6.3f", probwet_dry[i]
"\n"
(monthly values for alpha parameter)
"%6.3f", alpha[i]
"\n"
(monthly values for beta parameter)
"%6.3f", beta[i]
"\n"
(temperature and solar rad. parameters)
"%8.2f%8.2f%8.2f%8.2f\n", wdparams[0],wdparams[1],wdparams[2],
   wdparams[3]
"%8.2f\n", wdparams[4]
"%8.2f%8.2f%8.2f%8.2f\n", wdparams[5],wdparams[6],wdparams[7],
   wdparams[8]
"%8.2f%8.2f\n", wdparams[9],wdparams[10]
"%8.2f\n", wdparams[11]
```

Example:

```
[WEATHER INFO FOR WGEN FROM TABLES FOR CORVALLIS, OREGON]

  12    1 44.0    0    0
0.791 0.728 0.750 0.638 0.611 0.555 0.404 0.494 0.507 0.659 0.776 0.755
0.411 0.341 0.293 0.304 0.215 0.151 0.045 0.086 0.148 0.233 0.339 0.427
0.866 0.763 0.964 0.867 0.998 0.776 0.826 0.829 0.722 0.866 0.833 0.827
0.435 0.380 0.270 0.198 0.172 0.221 0.148 0.157 0.296 0.337 0.380 0.434
  65.00   24.00    0.14   -0.06
  59.00
  40.00   13.00    0.17   -0.08
 420.00  302.00
 267.00
```

The WGEN daily summary file. (Note: every fourth year contains a daily summary for Julian date 366.)

Format:

```
(j is the Julian date of the present day being printed)
"%5d%5d%5d%5d%7.2f%7.0f%7.0f%7.0f\n", month,day,year,j,
   Daily_rain[j],Daily_maxt[j],Daily_mint[j],Daily_rad[j]
```

Example:

```
   3   26    1   85   0.03    40    26    269
   3   27    1   86   0.29    43    29    179
   3   28    1   87   0.26    46    28    287
   3   29    1   88   0.33    43    31    309
   3   30    1   89   0.53    41    29    200
   3   31    1   90   0.90    52    27    485
   4    1    1   91   0.00    46    26    571
   4    2    1   92   0.23    39    25    536
   4    3    1   93   0.00    46    28    404
   4    4    1   94   0.01    50    35    224
   4    5    1   95   0.11    62    38    374
```

The WGEN yearly summary file contains a summary for each year generated. Additionally, at the end of the file, is a summary with the same format as the yearly summaries, but the monthly values are averages over all of the years generated (this last summary is not used in the data manager program).

Format:

```
"SUMMARY FOR YEAR %d\n", year
"  Jan   Feb   Mar   Apr   May   Jun   Jul "
"  Aug   Sep   Oct   Nov   Dec   Year\n\n"
(monthly total wet days and year total)
"WET DAYS\n"
"%6d", yr_monthly_wetd[j]
"%6d", yr_total_wetd
(monthly total rainfall and year total)
"\nRAINFALL\n"
"%6.1f", yr_monthly_rain[j]
"%6.1f", yr_total_rain
(monthly ave. max temperature and year ave.)
"\nAVE MAX TEMP\n"
"%6.1f", yr_monthly_maxt[j]
"%6.1f", yr_ave_maxt
(monthly ave. min temperature and year ave.)
"\nAVE MIN TEMP\n"
"%6.1f", yr_monthly_mint[j]
"%6.1f", yr_ave_mint
(monthly ave. daily total solar radiation and year ave.)
"\nAVE RAD\n"
"%6.1f", yr_monthly_rad[j]
"%6.1f", yr_ave_rad
"\n\n"
```

Example:

```
SUMMARY FOR YEAR 10
  Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec   Year

WET DAYS
    22    15    14    16     8     5     6     5     4    11    22    18   146
RAINFALL
   7.3   3.9   2.5   2.7   1.8   0.4   0.5   0.4   1.9   2.8   8.5   4.9  37.4
AVE MAX TEMP
  38.7  41.2  46.2  63.8  73.3  81.3  89.9  84.6  73.5  61.9  49.3  37.7  61.8
AVE MIN TEMP
  27.8  29.1  30.2  44.4  47.0  47.3  55.1  50.7  45.6  40.9  34.1  28.6  40.1
AVE RAD
  86.3 173.3 338.8 427.1 614.7 706.4 654.1 558.2 399.4 222.8  77.9  61.4 360.0
```

Variables used in weather data manager program

int	arrayid: ID number for hourly, daily, or precip occurrance summary
int	ascii_number: ASCII value of the user's choice
float	*ave_gen_temp: address of the ave temp.
FILE	*backup_fp: pointer to file of back-up data
FILE	*current_fp: pointer to file of current IBSNAT-formatted data
int	*calendar_day: address to receive the calendar day
int	*calendar_month: address to receive the calendar month
int	calculated_jdate: Julian date calculated from calendar date
float	checkvalue: value to check
int	checkvalue_month: month the value was collected
int	choice: user's choice
int	chosen_year: listed number of the gen'd year chosen by user
char	chosen_year_string[5]: string to get the listed number of the gen'd year chosen by user
float	converted_newvalue: new value converted to proper units for QC tests
float	converted_value: value converted to proper units for QC tests
struct formatted_day	current_daydata: IBSNAT-format daily summary from current file
FILE	*dailysum_fp: pointer to WGEN daily summary file
struct day	*daydata_ptr: address of data of a daily summary
int	*difference: address of difference between present and previous Julian dates
char	date_line[11]: string to read new date entered by user
int	day: day of month
struct day	day_data[DAYS_PREVIOUS]: present and previous daily summaries
int	day_for_showing: used to show date of hourly value that failed QC
int	day_from_file: day read from IBSNAT-formatted file
int	day_to_edit: Julian date of daily summary that user wants to edit
int	day_to_edit_found: indicates if the daily summary that the user wants to edit has been found in the current data file
static int	days_per_month: number of days in each month
int	days_per_year: number of days in the year to convert
int	difference: difference between the present day's Julian date and the previous day's Julian date
int	divisor_for_averages: number of months used to calculate ave. temp.
float	dummy_float: used in a function call when not all parameters are used
char	dummy_string[2]: used to read carriage return at end of line
char	*error_code[NUMBER_OF_ERROR_CODES]: (global) QC error code descriptions
int	edit_course: signifies course user will follow while editing program info

```
char                    entered_string[9]: used to get value
                           entered by user
float                   entered_value: numeric value from the
                           string entered by user
int                     erroractions[VALUES_WITH_ERRORS][ONE_
                           ERROR_ONE_ACTION]: array of QC errors
                           and actions
char                    *file_name_to_open: address of the name of
                           file to open
char                    *file_name_to_view: address of name of
                           file to view
char                    *file_open_activity: address of how to
                           open a file (read, write, append)
int                     *first_day_year: address of the year of
                           the first daily summary collected
struct formatted_day    *formatted_ptr: address of the formatted
                           data
FILE                    *fp: pointer to a file
char                    file_line[15]: used to get file names from
                           user
int                     finished: flag indicating when user is
                           finished editing
float                   first_array[YRS_GEN_DATA]: array of precip
                           or temp summaries to sort by
int                     first_gen_day: first day of generated data
                           to be put in current/gen'd file
int                     first_gen_day_month: month of first day
                           of gen'd data put in current/gen'd file
int                     first_gen_day_year: year of first day of
                           generated data to be put in current/
                           gen'd file
int                     first_time_through: indicates the first
                           time through Julian date QC tests
struct formatted_day    formatted_data: daily summary in IBSNAT
                           format
struct gen_data         *gendata_ptr: address of a day of gen'd
                           data
float                   gen_maxt_total: used to calculate ave.
                           temp.
float                   gen_mint_total: used to calculate ave.
                           temp.
float                   gen_precip[YRS_GEN_DATA]: total precip for
                           each gen'd year
float                   gen_temp[YRS_GEN_DATA]: ave. temp for each
                           gen'd year
int                     gen_year_end: number of days in the year
                           of gen'd data chosen by user
int                     gen_year_number: number of the year of
                           gen'd data chosen by user
struct gen_data         gendata: gen'd data for a day
int                     generated_year[YRS_GEN_DATA]: array of the
                           year numbers of the gen'd data
struct hour             *hourdata_ptr: address of data of an
                           hourly summary
int                     hour: hour in 24-hour format
struct hour             hour_data[HOURS_PREVIOUS]: present and
                           previous hourly summaries
FILE                    *info_fp: pointer for program info file
int                     i: loop counter
char                    id_line[5]: used to get ID's from user
struct info_codes       info: (global) program information
int                     j: loop counter
int                     *jdate_action: address of QC action code
                           from Julian date tests
int                     *jdate_error: address of QC error code
                           from Julian date tests
int                     jdate_action: QC action code (if any)
                           resulting from Julian date QC tests
int                     jdate_error: QC error code (if any)
                           resulting from Julian date QC tests
int                     jdate_to_convert: Julian date to convert
int                     k: loop counter
int                     keeping_newvalue: indicates if user wants
```

```
                    to keep or re-enter the new value that
                    failed QC tests
int                 last_current_day: last day of current data
                    in the current/gen'd data file
int                 last_current_day_year: year of the last
                    day of current data in the current/gen'd
                    data file
int                 last_gen_day: last day of generated data
                    to be put in current/gen'd file
int                 last_gen_day_month: month of last day
                    of gen'd data put in current/gen'd file
int                 last_gen_day_year: year of last day of
                    generated data to be put in current/
                    gen'd file
float               latitude: latitude of weather station (for
                    IBSNAT format)
char                latlon_line[9]: used to get latitude and
                    longitude from user
int                 line_count: used to determine when screen
                    is full of data
char                line_from_file[100]: string to read a line
                    of data from file
char                location_id[3]: string for location ID (for
                    IBSNAT format)
float               longitude: longitude of weather station
                    (for IBSNAT format)
char                *maxmin[MAXMIN]: labels for maximum and
                    minimum
char                *measurement: address of string containing
                    name of the current QC measurement
char                *measurement_units: address of string
                    containing units of the current QC msrmt
int                 *missingday_jdate: address of the previous
                    day's date (becomes missing day's date)
struct day          *missingday_ptr: address of missing daily
                    summary data
int                 *missingday_year: address of the previous
                    day's year (becomes missing day's year)
char                *month_name[MONTHS+1]: (global) month names
float               maxchange: maximum allowable change
struct day          missingday: data for missing daily summary
int                 missingday_jdate: previous day's date
                    (becomes missing day's date)
int                 missingday_year: previous day's year
                    (becomes missing day's year)
int                 month: number of the month (1-12)
char                *month_day[8]: string for entered month
                    and day
int                 month_for_showing: used to show date of
                    hourly value that failed QC
char                month_from_file[4]: month name read from
                    IBSNAT-formatted file
int                 new_julian_date: Julian date of the new
                    date entered by user
float               new_latlon: new value for latitude or
                    longitude
float               newvalue: new value entered by user for
                    one which failed QC
int                 number_to_sort: number of values to sort
int                 *open_result: address of flag signifying
                    if error occurred during file opening
char                old_qc_file[13]: name of old QC file;
                    used to rename file to new name
int                 open_result: signifies if error occurred
                    during file opening
float               opposite_value: opposite value for
                    comparison
int                 *precip_time: address of time of precip
                    occurrance
int                 *precip_amount: address of amount of
                    precip occurrance
struct day          *presentday_ptr: address of the daily
                    summary data for the present day
```

```
int             *presentday_year: address of the year of
                   the present data set being run through
                   QC
struct hour     *present_hour_ptr: address of data for the
                   present hour being run through QC
FILE            *previous_fp: pointer to file of previous
                   data
struct hour     *previous1hour_ptr: address of the
                   collected hourly summary data for the
                   hour previous to the present hour
struct hour     *previous2hour_ptr: address of the
                   collected hourly summary data for two
                   hours previous to the present hour
struct day      *previousday_ptr: address of the collected
                   daily summary data for the day previous
                   to the present day being run through QC
float           pardat: flag indicating if PAR is included
                   in weather file (for IBSNAT format)
float           parfac: factor for converting MJ/m2 to PAR
                   (for IBSNAT format)
int             precip_amount: amount of a precipitation
                   occurrance
int             precip_time: time of a precipitation
                   occurrance
int             presentday_day: day of the present daily
                   summary
int             presentday_jdate: Julian date of the
                   present day's data
int             presentday_month: month of the present
                   daily summary
int             presenthour_hour: hour of the present
                   hourly summary
int             presenthour_month: month of the present
                   hourly summary
float           presenthour_rh: rh value of the present
                   hourly summary
float           presenthour_sr: solar rad. value of the
                   present hourly summary
float           presenthour_temp: temperature value of the
                   present hourly summary
int             previous1hour_hour: hour of the previous
                   hourly summary
float           previous1hour_rh: rh value of the previous
                   hourly summary
float           previous1hour_temp: temperature of the
                   previous hourly summary
float           previous1value: immediately previous value
                   for comparison
float           previous2hour_rh: rh value of the two-hour
                   previous summary
float           previous2hour_temp: temperature of the
                   two-hour previous summary
float           previous2value: next previous value for
                   comparison
int             previousday_day: day of the previous
                   daily summary
int             previousday_jdate: Julian date of the
                   previous day's data
int             previousday_month: month of the previous
                   daily summary
int             previousday_year: year of the previous
                   day's data
int             print_result: signifies if error occurred
                   while printing to a file
FILE            *qc_fp: pointer to QC values file
char            *qc_measurement[6]: string array of names
                   of the measurements that have QC values
struct qc       *qcptr: address of QC values
int             qc_action_from_file: a QC action read from
                   IBSNAT-formatted file
float           qc_data[MAXMIN]: array with max and min
                   values in it
int             qc_error_from_file: a QC error read from
```

```
                    IBSNAT-formatted file
struct qc           qcvalues: values for QC tests
FILE                *rawdata_fp: pointer to file of raw data
int                 rungen_result: signifies if WGEN ran with
                       or without an error occurring
int                 scan_result: signifies if any error/
                       actions are recorded
int                 screen_view: signifies if user is viewing
                       file on screen or printer
float               second_array[YRS_GEN_DATA]: array of
                       precip or temp summaries; follows
                       sorting of first array
long                start_year_pos: location in WGEN daily
                       summary file where the chosen year of
                       gen'd data begins
char                station_id[3]: string for station ID (for
                       IBSNAT format)
char                station_name_line[9]: used to get station
                       name from user
FILE                *temporary_fp: pointer to temporary file
float               *total_gen_precip: address of total precip
float               temporary_value: used to hold a value that
                       failed QC so user may change it
float               temporary1: holds value being sorted
float               temporary2: holds second value being
                       sorted
int                 temporary3: holds month number being
                       sorted
int                 third_array[YRS_GEN_DATA]: array of year
                       numbers; follows sorting of first array
struct day          *unformatted_ptr: address of the daily
                       data to be formatted
char                *units_code[11]: (global) units for values
int                 unit_code: code signifying units for a
                       value
int                 use_all_generated_data: signifies if user
                       wants only gen'd data in the current/
                       gen'd file
char                *value_code[NUMBER_OF_VALUE_CODES]:
                       (global) value descriptions
float               *value_to_fix: address of value that
                       failed QC
FILE                *view_in_fp: pointer to data file to view
FILE                *view_out_fp: pointer to output file for
                       viewing (screen or printer)
int                 val_code: number signifying what the
                       value represents
int                 value_to_change: user's choice of which
                       daily summary value to change
float               value_to_convert: value to be converted to
                       proper units for QC tests
int                 view_or_edit: indicates user's choice to
                       view or edit a file
struct wgen_input   *wgen_input_ptr: address of data for
                       WGEN input file
char                *wgen_input_string[12]: string used to
                       get value for WGEN input file
float               *wgen_input_value: address of value for
                       WGEN input file
FILE                *wgeninput_fp: pointer to WGEN input
                       parameters file
int                 warning: number indicating type of QC
                       error found (if any)
float               warning_value: value which failed QC
char                wgen_entered_string[12]: string to get WGEN
                       input value from user
struct wgen_input   wgen_input_data: data for the WGEN input
                       file
char                wgen_input_desc[81]: string to get comment
                       line for WGEN input file from user
FILE                *yearlysum_fp: pointer to WGEN yearly
                       summary file
int                 year: two-digit year
```

```
int             year_for_showing: used to show date of
                   hourly value that failed QC
char            year_line[5]: used to get year data
                   collection started from user
char            year_string[3]: year data collection
                   starts; used to make file names
int             year_to_convert: year of the Julian date
                   to convert
float           yearlysum_maxt: max. temp. value read
                   from WGEN yearly summary file
float           yearlysum_mint: min. temp. value read
                   from WGEN yearly summary file
float           yearlysum_precip: precip value read from
                   WGEN yearly summary file
```

PROGRAM LISTING OF

THE WEATHER DATA MANAGER PROGRAM

written by

William S. Donaldson

WMAIN.C

```
/*************************************************************************/
/* Weather Data Manager Program                                          */
/*    Created by William Donaldson   1991                                */
/*                                                                       */
/*    Other parts of the program are contained in:  WQC.C                */
/*                                                  WEDIT.C              */
/*                                                  WGENER.C             */
/*                                                  WMISC.C              */
/*                                                  WHDR.H               */
/*                                                  WGLOBAL.H            */
/*************************************************************************/


/* Include files */
#include <conio.h>      /* for console functions */
#include <stdio.h>      /* for file functions */
#include <graph.h>      /* for clear-screen function only */
#include <process.h>    /* for exit function */
#include <string.h>     /* for string copy function */
#include "whdr.h"       /* defines, structs, functions for this program */

/* Initialization of global variables */

/* Array of value code descriptions */
char *value_code[NUMBER_OF_VALUE_CODES] =
   /* 0*/{"Day [Hourly]",
   /* 1*/ "Hour [Hourly]",
   /* 2*/ "Hourly Ave. Temp.",
   /* 3*/ "Hourly Rel. Humidity",
   /* 4*/ "Hourly Total Solar Rad.",
   /* 5*/ "Hourly Ave. Wind Speed",
   /* 6*/ "Hourly Ave. Wind Dir.",
   /* 7*/ "Hourly Std. Dev. of Wind Dir.",
   /* 8*/ "Day [Daily]",
   /* 9*/ "Hour [Daily]",
   /*10*/ "Daily Ave. Temp.",
   /*11*/ "Daily Max. Temp.",
   /*12*/ "Daily Min. Temp.",
   /*13*/ "Daily Max. Rel. Humidity",
   /*14*/ "Daily Min. Rel. Humidity",
   /*15*/ "Daily Total Solar Rad.",
   /*16*/ "Daily Max. Wind Speed",
   /*17*/ "Daily Ave. Wind Speed",
   /*18*/ "Daily Total Precip.",
   /*19*/ "Daily Datalogger Moisture",
   /*20*/ "Daily Battery voltage",
   /*21*/ "Daily Datalogger Max. Temp.",
   /*22*/ "Daily Datalogger Min. Temp.");

/* Array of error code descriptions */
char *error_code[NUMBER_OF_ERROR_CODES] =
   {"not used", "not used",
   /*DATEMISSING*/  "Missing one or more daily summaries prior to this date",
   /*DATEOUTRNG*/   "The date for this data is not valid",
   /*DATEOUTORDR*/  "The date for this daily summary is out of sequence",
   /*FILLEDDATA*/   "Data for this date was missing",

   /*MAXTOUTRNG*/   "Max. temperature too high or too low",
   /*MAXTCONSTNT*/  "Max. temperature constant over time",
   /*MAXTCHANGE*/   "Max. temp. change is greater than allowable",
   /*MAXTLESSMINT*/ "Max. temp. is less than min. temp.",
   /*MAXTEDITED*/   "Max. temp. was edited",

   /*MINTOUTRNG*/   "Min. temperature too high or too low",
   /*MINTCONSTNT*/  "Min. temperature constant over time",
   /*MINTCHANGE*/   "Min. temp. change is greater than allowable",
   /*MINTEDITED*/   "Min. temp. was edited",

   /*SOLAROUTRNG*/  "Total solar rad. too high or too low",
   /*SOLAREDITED*/  "Total solar rad. was edited",
```

```
   /*PRECIPOUTRNG*/ "Total precip. too high or too low",
   /*PRECIPEDITED*/ "Total precip. was edited",

   /*AVETOUTRNG*/   "Ave. temperature too high or too low",
   /*AVETCONSTNT*/  "Ave. temperature constant over time",
   /*RHOUTRNG*/     "Relative humidity too high or too low",
   /*RHCONSTNT*/    "Relative humidity constant over time",
   /*MXRHLESSMNRH*/ "Max. RH is less than min. RH",
   /*WINDCONSTNT*/  "Max. wind speed constant over time",
   /*SOLARPOSITIV*/ "Positive solar rad. at night",
   /*BATTRYOUTRNG*/ "Battery voltage too high or too low",
   /*MOISTROUTRNG*/ "Moisture in datalogger is too high or too low",
   /*HOURMISSING*/  "Missing line of hourly data (may cause errant daily data)",
};

/* Array of unit code descriptions */
char *units_code[11] =
   /* 0*/{"(degrees C)",
   /* 1*/ "(cm)",
   /* 2*/ "(MJ/m2/day)",
   /* 3*/ "(kW/m2)",
   /* 4*/ "(degrees F)",
   /* 5*/ "(in)",
   /* 6*/ "(miles/hr)",
   /* 7*/ "(%)",
   /* 8*/ "(index)",
   /* 9*/ "(volts)",
   /*10*/ "",};

char *month_name[MONTHS+1] = {"not used",
                              "Jan",
                              "Feb",
                              "Mar",
                              "Apr",
                              "May",
                              "Jun",
                              "Jul",
                              "Aug",
                              "Sep",
                              "Oct",
                              "Nov",
                              "Dec"};

struct info_codes info;


/* MAIN PROGRAM */

main()
/*---------------------------------------------------------------------------*/
/* Prints main menu, gets the user's choice of what to do, and calls the   */
/*  appropriate function.  Loads program information.                        */
/* Receives: none.                                                           */
/* Globals: none.                                                            */
/* Returns: value of 1 if program exits properly                             */
/* Calls: ShowMainMenu                                                       */
/*        RunQC                                                              */
/*        GetGeneratedData                                                   */
/*        FileViewEdit                                                       */
/*        MiscFunctions                                                      */
/*        LoadProgramInfo                                                    */
/*        EditProgramInfo                                                    */
/*---------------------------------------------------------------------------*/
{
int choice;  /* user's choice */

int LoadProgramInfo(void);
void ShowMainMenu(void);
void RunQC(void);
void FileViewEdit(void);
void GetGeneratedData(void);
void MiscFunctions(void);
```

```
   /* Print welcome */
   _clearscreen(_GCLEARSCREEN);
   printf("\n\n\n\t\tWELCOME TO THE WEATHER DATA MANAGER\n\n\n");
   printf("\n\t\t    (press any key to continue)");
   getch();

   /* Get program info from file; make user edit info if none exists */
   if(LoadProgramInfo() == ERROR)
      EditProgramInfo(NO_INFO_FILE);

   while(TRUE){
      /* Show menu and get user's choice */
      _clearscreen(_GCLEARSCREEN);
      ShowMainMenu();
      choice = getche();
      /* Break out of loop if user chooses quit */
      if(choice == '5')
         break;
      /* Call appropriate function based on user's choice */
      switch(choice){
      case '1':
         RunQC();
         break;
      case '2':
         FileViewEdit();
         break;
      case '3':
         GetGeneratedData();
         break;
      case '4':
         MiscFunctions();
         break;
      default:
         printf("\n\n*** Enter 1-5 ***\n");
         printf("\n(press any key)");
         getch();}}
   return(1);
}  /* end main */


void ShowMainMenu(void)
/*-------------------------------------------------------------------------*/
/* Prints main menu to screen.                                             */
/* Receives: none.                                                         */
/* Globals:  none.                                                         */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   printf("\n\n\n\t\t\tMAIN MENU FOR WEATHER DATA MANAGER\n\n\n");
   printf("\t\t1.  Retrieve Data, Run QC, Add Data to IBSNAT File\n\n");
   printf("\t\t2.  View/Edit Files\n\n");
   printf("\t\t3.  Run Weather Generator\n\n");
   printf("\t\t4.  Miscellaneous Functions\n\n");
   printf("\t\t5.  Quit\n\n\n\n");
   printf("\t\tYour choice:  ");
}  /* end ShowMainMenu */


int LoadProgramInfo(void)
/*-------------------------------------------------------------------------*/
/* If program info file exists, info is retrieved from file.               */
/* Receives: none.                                                         */
/* Globals:  none.                                                         */
/* Returns: OK if info is loaded                                           */
/*          ERROR if info file does not exist                              */
/* Calls: ScanProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
{
FILE *info_fp;     /* pointer to program info file */
int open_result;   /* signifies if error occured during file opening */
int ScanProgramInfo(FILE *);
```

```
    /* Open file and check if file doesn't exist */
    info_fp = OpenFile(&open_result, INFO_FILE, READ);
    if(open_result == ERROR)
       return(ERROR);
    /* Load data into global struct; stop if insufficient data in file */
    if(ScanProgramInfo(info_fp) == FILE_END){
       fclose(info_fp);
       return(ERROR);}
    fclose(info_fp);
    /* Return OK to show data is loaded */
    return(OK);
}  /* end LoadProgramInfo */


int ScanProgramInfo(FILE *info_fp)
/*-------------------------------------------------------------------------*/
/* Gets data from program info file and puts it into global struct.        */
/* Receives:  *info_fp:  pointer to program info file                      */
/* Globals: info                                                           */
/* Returns:  OK if data successfully loaded                                */
/*           FILE_END if inadequate data in file                           */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
char dummy_string[2]; /* used to read carriage return at end of line */

    /* Get station name */
    if(fscanf(info_fp, "%[^\n]", info.sta_name) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    /* Get file names */
    if(fscanf(info_fp, "%[^\n]", info.raw_file) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    if(fscanf(info_fp, "%[^\n]", info.prev_file) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    if(fscanf(info_fp, "%[^\n]", info.backup_file) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    if(fscanf(info_fp, "%[^\n]", info.current_file) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    if(fscanf(info_fp, "%[^\n]", info.curgen_file) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    if(fscanf(info_fp, "%[^\n]", info.qc_file) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    /* Get institute and station id */
    if(fscanf(info_fp, "%[^\n]", info.location_id) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    if(fscanf(info_fp, "%[^\n]", info.station_id) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    fscanf(info_fp, "%[\n]", dummy_string);
    /* Get latitude, longitude */
    if(fscanf(info_fp, " %f %f ", &info.lat, &info.lon) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    if(fscanf(info_fp, " %d ", &info.year) == EOF)
       /* If at end of file already, return error */
       return(FILE_END);
    /* Set unchangable values */
```

```
   info.parfac = 12.07;
   info.pardat = 0.0;

   return(OK);
}  /* end ScanProgramInfo */



/* Global Functions */

int CalDate(int jdate_to_convert, int year_to_convert, int *calendar_month,
      int *calendar_day)
/*-------------------------------------------------------------------------*/
/* Converts a julian date to a calendar date.                              */
/* Receives:  jdate_to_convert:  Julian date to convert                    */
/*            year_to_convert:  year of the Julian date to convert         */
/*            *calendar_month:  address to receive the calendar month      */
/*            *calendar_day:  address to receive the calendar day          */
/* Globals:  none.                                                         */
/* Returns:  CD_SUCCESS if date successfully converted                     */
/*           CD_INVALID_JDATE if julian date is invalid                    */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
static int days_per_month[12] = { 31, 28, 31, 30, 31, 30,
                                  31, 31, 30, 31, 30, 31 };
int days_per_year = 365;

   /* If a leap year, change Feb days to 29 */
   if ((year_to_convert % 4) == 0){
      days_per_month[1] = 29;
      days_per_year   += 1;}
   else
      days_per_month[1] = 28;

   /*-- check for invalid jdate --*/
   if ((jdate_to_convert < 1) || (jdate_to_convert > (366)))
      return( CD_INVALID_JDATE );

   /* initialize month and day */
   *calendar_month = 1;
   *calendar_day = 31;
   /* Find proper month */
   while (*calendar_day < jdate_to_convert)
      *calendar_day += days_per_month[(*calendar_month)++];
   /* Get proper day */
   *calendar_day = jdate_to_convert - *calendar_day +
         days_per_month[*calendar_month-1];
   return( CD_SUCCESS );
}  /* end CalDate */


int CheckChange(float checkvalue, float previous1value, float maxchange)
/*-------------------------------------------------------------------------*/
/* Checks a value to see if it has changed more than is allowable.         */
/* Receives:  checkvalue:  value to check                                  */
/*            previous1value:  immediately previous value for comparison   */
/*            maxchange:  the maximum allowable change (float)             */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value is less than the allowable change           */
/*           FAIL if the value is greater                                  */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
   /* Check value to see if value is not > previous1value +- maxchange */
   if((checkvalue <= previous1value + maxchange) &&
         (checkvalue >= previous1value - maxchange))
      return(PASS);
   else
      return(FAIL);
}  /* end CheckChange */
```

```
int CheckConstant(float checkvalue, float previous1value, float previous2value)
/*-------------------------------------------------------------------------*/
/* Checks a value to see if it is the same as the past one or two values.  */
/* Receives:  checkvalue:  value to check                                  */
/*            previous1value:  immediately previous value for comparison   */
/*            previous2value:  next previous value for comparison          */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value is not the same as the past value(s)        */
/*           FAIL if the value is the same                                 */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
   /* Check value against both past values if data is given for both */
   if(previous2value < (float) NO_DATA){
      /* See if value is not equal to past value */
      if(checkvalue != previous1value)
         return(PASS);
      else
         /* If value = past value, see if it is not equal to next past value */
         if(checkvalue != previous2value)
            return(PASS);
         else
            return(FAIL);}
   /* Just check the immediate past value if no data is given for next past */
   else{
      /* See if value is not equal to past value */
      if(checkvalue != previous1value)
         return(PASS);
      else
         return(FAIL);}
}  /* end CheckConstant */


int CheckExtreme(float checkvalue, float maxmin[MAXMIN])
/*-------------------------------------------------------------------------*/
/* Determines if a value is within a range from min to max.                */
/* Receives:  checkvalue:  value to check                                  */
/*            maxmin[]:  array with max and min values in it               */
/* Globals:  none.                                                         */
/* Returns:  PASS if value is within range                                 */
/*           FAIL if value is out of range                                 */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
   /* Check to see if value is within range of min to max */
   if((checkvalue >= maxmin[MIN]) && (checkvalue <= maxmin[MAX]))
      return(PASS);
   else
      return(FAIL);
}  /* end CheckExtreme */


int CheckMaxMin(float checkvalue, float opposite_value)
/*-------------------------------------------------------------------------*/
/* Checks a value to see if it is > or < its opposite value.               */
/* Receives:  checkvalue:  value to check                                  */
/*            opposite_value:  the opposite value for comparison           */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value is > the opposite value                     */
/*           FAIL if the value is < the opposite value                     */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
   /* Check to see if value is > the opposite value */
   if(checkvalue > opposite_value)
      return(PASS);
   else
      return(FAIL);
}  /* end CheckMaxMin */


int DayPrecQC(float checkvalue, int checkvalue_month, struct qc *qcptr)
/*-------------------------------------------------------------------------*/
```

```
/* Checks a precipitation value to see if it is reasonable.                */
/* Receives:  checkvalue:  value to check                                  */
/*            checkvalue_month:  month the value was collected             */
/*            *qcptr:  address of qc values                                */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value passes all qc tests                         */
/*           FAIL if the value fails any one qc test                       */
/* Calls:  CheckExtreme                                                    */
/*-------------------------------------------------------------------------*/
{
   /* Make sure precip is within range (different values for each month) */
   if(CheckExtreme(checkvalue, qcptr->prec[checkvalue_month]) != PASS)
         return(PRECIPOUTRNG);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end DayPrecQC */


int DaySRQC(float checkvalue, int checkvalue_month, struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Checks a solar radiation value to see if it is reasonable.              */
/* Receives:  checkvalue:  value to check                                  */
/*            checkvalue_month:  month the value was collected             */
/*            *qcptr:  address of qc values                                */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value passes all qc tests                         */
/*           FAIL if the value fails any one qc test                       */
/* Calls:  CheckExtreme                                                    */
/*-------------------------------------------------------------------------*/
{
   /* Make sure SR is within range (different values for each month) */
   if(CheckExtreme(checkvalue, qcptr->sr[checkvalue_month]) != PASS)
      return(SOLAROUTRNG);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end DaySRQC */


int DayTempQC(float checkvalue, int checkvalue_month, struct qc *qcptr,
      float previous1value, float opposite_value, int val_code)
/*-------------------------------------------------------------------------*/
/* Checks a temperature value to see if it is reasonable.                  */
/* Receives:  checkvalue:  value to check                                  */
/*            checkvalue_month:  month the value was collected             */
/*            *qcptr:  address of qc values                                */
/*            previous1value:  immediately previous value for comparison   */
/*            opposite_value:  opposite value for comparison               */
/*            val_code:  number signifying what the value represents       */
/* Globals:  none.                                                         */
/* Returns:  PASS if the value passes all qc tests                         */
/*           FAIL if the value fails any one qc test                       */
/* Calls:  CheckExtreme                                                    */
/*         CheckConstant                                                   */
/*         CheckChange                                                     */
/*         CheckMaxMin                                                     */
/*-------------------------------------------------------------------------*/
{
   /* Make sure temp is within range (different values for each month) */
   if(CheckExtreme(checkvalue, qcptr->temp[checkvalue_month]) != PASS){
      if(val_code == VALDMXT)
         return(MAXTOUTRNG);
      else
         return(MINTOUTRNG);}
   /* Make sure temp is not exactly the same as yesterday's */
   if(previous1value < (float) NO_DATA){
      if(CheckConstant(checkvalue, previous1value, (float) NO_DATA) != PASS){
         if(val_code == VALDMXT)
            return(MAXTCONSTNT);
         else
            return(MINTCONSTNT);}
      /* Make sure temp has not undergone too much change from yesterday's */
      if(CheckChange(checkvalue, previous1value, qcptr->maxtempchng) != PASS){
         if(val_code == VALDMXT)
```

```
            return(MAXTCHANGE);
         else
            return(MINTCHANGE);}}
   /* Make sure max temp is not less than min temp */
   if((val_code == VALDMXT) && (opposite_value < (float) NO_DATA))
      if(CheckMaxMin(checkvalue, opposite_value) != PASS)
         return(MAXTLESSMINT);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end DayTempQC */


void FormatData(struct formatted_day *formatted_ptr,
      struct day *unformatted_ptr)
/*-------------------------------------------------------------------------*/
/* Converts daily data (deg. F, inches) to IBSNAT units (deg. C, cm), and  */
/*  puts the other info for the IBSNAT file in the converted struct.       */
/* Receives: *unformatted_ptr:  address of the daily data to be formatted  */
/*           *formatted_ptr:  address of the formatted data                */
/* Globals:  info                                                          */
/* Returns:  none.                                                         */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
   strcpy(formatted_ptr->location_id, info.location_id);
   strcpy(formatted_ptr->station_id, info.station_id);
   formatted_ptr->fyear = unformatted_ptr->year;
   formatted_ptr->fjdate = unformatted_ptr->jdate;
   /* Units of SR are already in MJ/m2/day, so don't change */
   formatted_ptr->fsr = unformatted_ptr->sr;
   /* Convert degrees F to degrees C for max and min temps ((F-32)*5/9) */
   formatted_ptr->fmaxt = (unformatted_ptr->maxt - FTOC32) * FTOC59;
   formatted_ptr->fmint = (unformatted_ptr->mint - FTOC32) * FTOC59;
   /* Convert inches to cm for precip */
   formatted_ptr->fprec = unformatted_ptr->prec * INTOCM;
   /* Make PAR data = 0 */
   formatted_ptr->fpar = 0.0;
}  /* end FormatData */


int GetFormattedDay(FILE *fp, struct formatted_day *formatted_ptr,
      int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION])
/*-------------------------------------------------------------------------*/
/* Reads a line of data in the IBSNAT format from the current data file,   */
/*  plus the month and day and any codes for qc errors and actions.        */
/* Receives: *fp:  pointer to file                                         */
/*           *formatted_ptr:  address of the formatted data                */
/*           erroractions[][]:  array of qc errors and actions             */
/* Globals: none.                                                          */
/* Returns: OK if not at end of file                                       */
/*          FILE_END if reach end of file                                  */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int scan_result;  /* signifies if any error/actions are recorded */
int i;  /* loop counter */
int j;  /* loop counter */
char month_from_file[4];  /* month name read from IBSNAT-formatted file */
int day_from_file;  /* day read from IBSNAT-formatted file */
int qc_error_from_file;  /* a QC error read from IBSNAT-formatted file */
int qc_action_from_file;  /* a QC action read from IBSNAT-formatted file */

   /* Set array of errors/actions to zero */
   for(i=0; i<=VALUES_WITH_ERRORS-1; i++)
      for(j=0; j<=ONE_ERROR_ONE_ACTION-1; j++)
         erroractions[i][j] = 0;

   /* Read in the data */
   if((fscanf(fp, "%2s%2s ", formatted_ptr->location_id,
         formatted_ptr->station_id)) != EOF){
      fscanf(fp, "%d %d %f %f %f %f %f", &formatted_ptr->fyear,
            &formatted_ptr->fjdate, &formatted_ptr->fsr,
            &formatted_ptr->fmaxt, &formatted_ptr->fmint,
```

```
            &formatted_ptr->fprec, &formatted_ptr->fpar);

      /* Read in the month and day */
      fscanf(fp, " %3s %d", month_from_file, &day_from_file);
      month_from_file[3] = '\0';

      /* Read in any error/actions and put in proper place in array */
      while(TRUE){
         scan_result = fscanf(fp, "  %d,%d", &qc_error_from_file,
               &qc_action_from_file);
         if((scan_result == 0) || (scan_result == EOF))
            return(OK);
         if(qc_error_from_file <= FILLEDDATA){
            erroractions[JDATE][QC_ERROR] = qc_error_from_file;
            erroractions[JDATE][QC_ACTION] = qc_action_from_file;}
         else if(qc_error_from_file <= MAXTEDITED){
            erroractions[MAXTEMP][QC_ERROR] = qc_error_from_file;
            erroractions[MAXTEMP][QC_ACTION] = qc_action_from_file;}
         else if(qc_error_from_file <= MINTEDITED){
            erroractions[MINTEMP][QC_ERROR] = qc_error_from_file;
            erroractions[MINTEMP][QC_ACTION] = qc_action_from_file;}
         else if(qc_error_from_file <= SOLAREDITED){
            erroractions[SOLAR][QC_ERROR] = qc_error_from_file;
            erroractions[SOLAR][QC_ACTION] = qc_action_from_file;}
         else{
            erroractions[PRECIP][QC_ERROR] = qc_error_from_file;
            erroractions[PRECIP][QC_ACTION] = qc_action_from_file;
            return(OK);}}}

   return(FILE_END);
}  /* end GetFormattedDay */


int GetGenDay(FILE *fp, struct gen_data *gendata_ptr)
/*-------------------------------------------------------------------------*/
/* Obtains a day of gen'd data from a file.                                */
/* Receives: *fp:  pointer to file                                         */
/*           *gendata_ptr:  address of a day of generated data             */
/* Globals: none.                                                          */
/* Returns: OK:  if data was read without problem                          */
/*          FILE_END:  if problem occured during read                      */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   if(fscanf(fp, "%d %d %d %d %f %f %f %f\n", &gendata_ptr->month,
         &gendata_ptr->day, &gendata_ptr->year, &gendata_ptr->jday,
         &gendata_ptr->prec, &gendata_ptr->maxt, &gendata_ptr->mint,
         &gendata_ptr->sr) != EOF)
      return(OK);
   return(FILE_END);
}  /* end GetGenDay */


int GetQCValues(struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Opens qc file (returns if file doesn't exist), gets qc data (returns if */
/*  insufficient data), closes file.                                       */
/* Receives:  *qcptr:  address of the qc values                            */
/* Globals:  info                                                          */
/* Returns:  OK if was able to get qc data                                 */
/*           ERROR if file doesn't exist or insufficient data              */
/* Calls:  OpenFile                                                        */
/*         ScanQC                                                          */
/*-------------------------------------------------------------------------*/
{
FILE *qc_fp;       /* pointer to QC file */
int open_result;   /* signifies if error occurred when opening file */
int ScanQC(FILE *, struct qc *);

   /* Open file and check if error occured */
   qc_fp = OpenFile(&open_result, info.qc_file, READ);
   if(open_result == ERROR)
      return(ERROR);
```

```
   /* Load qc data into qcptr; stop if insufficient data in file */
   if(ScanQC(qc_fp, qcptr) == FILE_END){
      fclose(qc_fp);
      return(ERROR);}
   fclose(qc_fp);
   /* Return OK to show qc data is in qcptr */
   return(OK);
}  /* end GetQCValues */


int JulDate(int month, int day, int year)
/*-------------------------------------------------------------------------*/
/* Converts month, day and year to a julian date.                          */
/* Receives:  month:  number of month (1-12)                               */
/*            day:  day of month                                           */
/*            year:  two-digit year                                        */
/* Globals:  none.                                                         */
/* Returns:  the julian date, if successful                                */
/*           JD_INVALID_MONTH if invalid month                             */
/*           JD_INVALID_DAY if invalid day                                 */
/*           JD_INVALID_YEAR if invalid year                               */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
int i;  /* loop counter */
int calculated_jdate = 0;  /* Julian date calculated from calendar date */
static int days_per_month[12] = { 31, 28, 31, 30, 31, 30,
                                  31, 31, 30, 31, 30, 31};

   /* If a leap year, change Feb days to 29 */
   if ((year % 4) == 0)
      days_per_month[1] = 29;
   else
      days_per_month[1] = 28;
   /* Trap errors in date */
   if ( (month < 1) || (month > 12) )
      return(JD_INVALID_MONTH);
   if ( (day   < 1) || (day > days_per_month[month-1]) )
      return(JD_INVALID_DAY);
   if ( (year  < 1) || ( year > 99) )
      return(JD_INVALID_YEAR);
   /* If month is Jan, the day is the same as the julian day */
   if (month == 1)
      return(day);
   /* Add days of each month to jdate until reach the month day is in */
   for (i = 0; i < (month-1); calculated_jdate += days_per_month[i++]);
   /* Add day to jdate to get final julian day */
   calculated_jdate += day;
   return(calculated_jdate);
}  /* end JulDate */


FILE *OpenFile(int *open_result, char *file_name_to_open,
      char *file_open_activity)
/*-------------------------------------------------------------------------*/
/* Opens a file for a given activity.                                      */
/* Receives:  *open_result:  address of flag if error occurred during file */
/*                           opening                                       */
/*            *file_name_to_open:  address of name of file to open         */
/*            *file_open_activity:  address of how to open a file (read,   */
/*                                  write, append                          */
/* Globals:  none.                                                         */
/* Returns:  file pointer to the opened file                               */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
FILE *fp;  /* pointer to file */

   /* Open file and set return value according to the result of that action */
   if((fp = fopen(file_name_to_open, file_open_activity)) == NULL)
      *open_result = ERROR;
   else
      *open_result = OK;
```

```
   /* Return file pointer to the file */
   return(fp);
}  /* end OpenFile */


void PrintDay(FILE *fp, struct day *daydata_ptr)
/*-------------------------------------------------------------------------*/
/* Prints daily data to backup file.                                       */
/* Receives: *fp: pointer to file                                          */
/*           *daydata_ptr:  address of the data of a daily summary         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   fprintf(fp, "%d,%d,", daydata_ptr->jdate, daydata_ptr->hour);
   fprintf(fp,"%.2f,%.2f,%.2f,%.2f,%.2f,", daydata_ptr->avet,
         daydata_ptr->maxt, daydata_ptr->mint, daydata_ptr->maxrh,
         daydata_ptr->minrh);
   fprintf(fp,"%.3f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n", daydata_ptr->sr,
          daydata_ptr->maxws, daydata_ptr->avews, daydata_ptr->prec,
          daydata_ptr->mois, daydata_ptr->volt, daydata_ptr->crmaxt,
          daydata_ptr->crmint);
   return;
}  /* end PrintDay */


void PrintDaysDataToScreen(struct day *daydata_ptr)
/*-------------------------------------------------------------------------*/
/* Prints the IBSNAT values of a day's data set to the screen.             */
/* Receives: *daydata_ptr:  address of data of a daily summary             */
/* Globals: value_code                                                     */
/*          units_code                                                     */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   printf("\t%6.2f %s  %s\n", daydata_ptr->maxt, units_code[FRNHT],
         value_code[VALDMXT]);
   printf("\t%6.2f %s  %s\n", daydata_ptr->mint, units_code[FRNHT],
         value_code[VALDMNT]);
   printf("\t%6.2f %s  %s\n", daydata_ptr->sr, units_code[MJPDA],
         value_code[VALDTSR]);
   printf("\t%6.2f %s  %s\n", daydata_ptr->prec, units_code[INCHS],
         value_code[VALDTPR]);
}  /* end PrintDaysDataToScreen */


void PrintFormattedDay(FILE *fp, struct formatted_day *formatted_ptr,
      int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION])
/*-------------------------------------------------------------------------*/
/* Prints IBSNAT data, the calendar date the data is for, and any error/   */
/*  action codes associated with the data to an IBSNAT formatted file.     */
/* Receives:  *fp:  pointer to file                                        */
/*            *formatted_ptr:  address of formatted daily data             */
/*            erroractions[][]:  array of qc errors and actions            */
/* Globals:  month_name                                                    */
/* Returns:  none.                                                         */
/* Calls:  CalDate                                                         */
/*-------------------------------------------------------------------------*/
{
int i;   /* loop counter */
int j;   /* loop counter */
int month;    /* month number (1-12) */
int day;      /* day of month */

   /* Print IBSNAT data to the file */
   fprintf(fp,"%2s%2s ", formatted_ptr->location_id, formatted_ptr->station_id);
   fprintf(fp, "%2d %3d %5.2f %5.1f %5.1f %5.1f %6.2f", formatted_ptr->fyear,
          formatted_ptr->fjdate, formatted_ptr->fsr, formatted_ptr->fmaxt,
          formatted_ptr->fmint, formatted_ptr->fprec, info.pardat);
   /* Get calendar day of the julian date; assume valid julian date */
   CalDate(formatted_ptr->fjdate, formatted_ptr->fyear, &month, &day);
```

```
   /* Print month name and day to the file */
   fprintf(fp, "  %s %2d", month_name[month], day);
   /* If data is generated, signify */
   if(erroractions[JDATE][QC_ERROR] == 'G')
      fprintf(fp, "  G");
   else
      /* If there are any error/action codes, print them to the file */
      for(i=0; i<=VALUES_WITH_ERRORS-1; i++)
         if(erroractions[i][QC_ERROR] != 0)
            fprintf(fp, "  %d,%d", erroractions[i][QC_ERROR],
                  erroractions[i][QC_ACTION]);
   /* Go to the next line of the file */
   fprintf(fp, "\n");
   return;
}  /* end PrintFormattedDay */


void PrintHour(FILE *fp, struct hour *hourdata_ptr)
/*-------------------------------------------------------------------------*/
/* Prints hourly data to the back-up file.                                 */
/* Receives: *fp: pointer to file                                          */
/*           *hourdata_ptr:  address of data of an hourly summary          */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   fprintf(fp, "%d,%d,", hourdata_ptr->jdate, hourdata_ptr->hour);
   fprintf(fp,"%.2f,%.2f,%.3f,%.2f,%.2f,%.2f\n", hourdata_ptr->temp,
      hourdata_ptr->rh, hourdata_ptr->sr, hourdata_ptr->ws,
      hourdata_ptr->wd, hourdata_ptr->wstd);
   return;
}  /* end PrintHour */


int QCNewValue(float checkvalue, float previous1value, int val_code,
      int checkvalue_month, struct qc *qcptr, float opposite_value)
/*-------------------------------------------------------------------------*/
/* Checks a new value entered by user to see if it is reasonable.          */
/* Receives:  checkvalue:  the value to check                              */
/*            previous1value:  immediately previous value for comparison   */
/*            val_code:  number signifying what the value represents       */
/*            checkvalue_month:  the month the value was collected         */
/*            *qcptr:  address of the qc values                            */
/*            opposite_value:  the opposite value for comparison           */
/* Globals:  error_code[]                                                  */
/* Returns:  PASS if the value passes all qc tests or if value fails a     */
/*           test and user wants to keep the value                         */
/*           FAIL if the value fails any qc test and user does not want to */
/*           keep the value                                                */
/* Calls:  DayTempQC                                                       */
/*         DaySRQC                                                         */
/*         DayPrecQC                                                       */
/*-------------------------------------------------------------------------*/
{
int warning;  /* number indicating type of QC error found (if any) */
int choice;   /* user's choice */

   /* Based on what value is being replaced, run qc on new value */
   switch(val_code){
      case VALDMXT:
      case VALDMNT:
         if((warning = DayTempQC(checkvalue, checkvalue_month, qcptr,
               previous1value, opposite_value, val_code)) == PASS)
            return(PASS);
         break;
      case VALDTSR:
         if((warning = DaySRQC(checkvalue, checkvalue_month, qcptr)) == PASS)
            return(PASS);
         break;
      case VALDTPR:
         if((warning = DayPrecQC(checkvalue, checkvalue_month, qcptr)) == PASS)
            return(PASS);
```

```
            break;}
      /* Print warning and see if user wants to keep the value */
      printf("NEW VALUE FAILED QC TESTS!\n");
      printf("   Reason:  %s\n", error_code[warning]);
      while(TRUE){
         printf("Keep this value?");
         choice = getche();
         printf("\n");
         if((choice == 'y') || (choice == 'Y'))
            return(PASS);
         if((choice == 'n') || (choice == 'N'))
            return(FAIL);}
   }  /* end QCNewValue */


   void SaveProgramInfo(void)
   /*-------------------------------------------------------------------------*/
   /* Saves program information to a file.                                    */
   /* Receives: none.                                                         */
   /* Globals:  info                                                          */
   /* Returns: none.                                                          */
   /* Calls: none.                                                            */
   /*-------------------------------------------------------------------------*/
   {
   FILE *info_fp;  /* pointer to program info file */
   int open_result;  /* indicates if error occurred during file opening */
   int print_result;  /* indicates if error occurred while printing to file */

      /* Open file */
      info_fp = OpenFile(&open_result, INFO_FILE, WRITE);
      if(open_result == ERROR){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}

      /* Save station name */
      if(fputs(info.sta_name, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      /* Save file names */
      if(fputs(info.raw_file, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      if(fputs(info.prev_file, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      if(fputs(info.backup_file, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      if(fputs(info.current_file, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      if(fputs(info.curgen_file, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      if(fputs(info.qc_file, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      /* Save institute and station id */
      if(fputs(info.location_id, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
      if(fputs(info.station_id, info_fp) == EOF){
         printf("\nError occured while saving information\n");
         exit(INFO_FILE_WRITE_ERROR);}
      fprintf(info_fp, "\n");
```

```
   /* Save latitude, longitude */
   print_result = fprintf(info_fp, "%6.2f %6.2f\n", info.lat, info.lon);
   if(print_result != 14){
      printf("\nError occured while saving information\n");
      exit(INFO_FILE_WRITE_ERROR);}
   print_result = fprintf(info_fp, "%2d\n", info.year);
   if(print_result != 3){
      printf("\nError occured while saving information\n");
      exit(INFO_FILE_WRITE_ERROR);}
   fclose(info_fp);
}  /* end SaveProgramInfo */


int ScanDay(FILE *fp, struct day *daydata_ptr)
/*-------------------------------------------------------------------------*/
/* Reads in line of daily data from the raw data file or previous file.    */
/* Receives: *fp: pointer to file                                          */
/*           *daydata_ptr:  address of data of a daily summary             */
/* Globals: none.                                                          */
/* Returns: OK if data was read                                            */
/*          FILE_END if error occurred                                     */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   if(fscanf(fp, "%d,%d,", &daydata_ptr->jdate, &daydata_ptr->hour)== EOF)
      return(FILE_END);
   if(fscanf(fp,"%f,%f,%f,%f,%f,", &daydata_ptr->avet, &daydata_ptr->maxt,
         &daydata_ptr->mint, &daydata_ptr->maxrh,
         &daydata_ptr->minrh) == EOF)
      return(FILE_END);
   if(fscanf(fp,"%f,%f,%f,%f,%f,%f,%f,%f\n", &daydata_ptr->sr,
         &daydata_ptr->maxws, &daydata_ptr->avews,
         &daydata_ptr->prec, &daydata_ptr->mois, &daydata_ptr->volt,
         &daydata_ptr->crmaxt, &daydata_ptr->crmint) == EOF)
      return(FILE_END);
   return(OK);
}  /* end ScanDay */


int ScanHour(FILE *fp, struct hour *hourdata_ptr)
/*-------------------------------------------------------------------------*/
/* Reads in line of hourly data from raw data file or previous file.       */
/* Receives: *fp: pointer to file                                          */
/*           *hourdata_ptr:  address of data of an hourly summary          */
/* Globals: none.                                                          */
/* Returns: OK if data was read                                            */
/*          FILE_END if error occurred                                     */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   if(fscanf(fp, "%d,%d,", &hourdata_ptr->jdate, &hourdata_ptr->hour)
         == EOF)
      return(FILE_END);
   if(fscanf(fp,"%f,%f,%f,%f,%f\n", &hourdata_ptr->temp,
         &hourdata_ptr->rh, &hourdata_ptr->sr, &hourdata_ptr->ws,
         &hourdata_ptr->wd, &hourdata_ptr->wstd) == EOF)
      return(FILE_END);
   return(OK);
}  /* end ScanHour */


int ScanQC(FILE *qc_fp, struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Gets data from QC file and puts it into QC struct.                      */
/* Receives:  *qc_fp:  pointer to qc file                                  */
/*            *qcptr:  address of qc data                                  */
/* Globals: none.                                                          */
/* Returns:  OK if qc data successfully loaded                             */
/*           FILE_END if inadequate data in file                           */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
int i;   /* loop counter */
```

```
int j = 0;   /* loop counter */
char line_from_file[100]; /* string to read a line of data from file */

   /* Get monthly temp max and mins */
   for(i=1; i<=MONTHS; i++){
      if(fgets(line_from_file, 60, qc_fp) == NULL)
         /* If at end of file already, return error */
         return(FILE_END);
      if(sscanf(line_from_file, " %f, %f ", &qcptr->temp[i][j],
            &qcptr->temp[i][j+1]) != 2)
         /* If at end of file already, return error */
         return(FILE_END);}

   /* Get monthly solar rad. max and mins */
   for(i=1; i<=MONTHS; i++){
      if(fgets(line_from_file, 60, qc_fp) == NULL)
         /* If at end of file already, return error */
         return(FILE_END);
      if(sscanf(line_from_file, " %f, %f ", &qcptr->sr[i][j],
            &qcptr->sr[i][j+1]) != 2)
         /* If at end of file already, return error */
         return(FILE_END);}

   /* Get monthly precip max and mins */
   for(i=1; i<=MONTHS; i++){
      if(fgets(line_from_file, 60, qc_fp) == NULL)
         /* If at end of file already, return error */
         return(FILE_END);
      if(sscanf(line_from_file, " %f, %f ", &qcptr->prec[i][j],
            &qcptr->prec[i][j+1]) != 2)
         /* If at end of file already, return error */
         return(FILE_END);}

   /* Get rel. humidity max and min */
   if(fgets(line_from_file, 60, qc_fp) == NULL)
      /* If at end of file already, return error */
      return(FILE_END);
   if(sscanf(line_from_file, " %f, %f ",&qcptr->rh[j],&qcptr->rh[j+1]) != 2)
      /* If at end of file already, return error */
      return(FILE_END);

   /* Get datalogger voltage max and min */
   if(fgets(line_from_file, 60, qc_fp) == NULL)
      /* If at end of file already, return error */
      return(FILE_END);
   if(sscanf(line_from_file, " %f, %f ",&qcptr->volt[j],&qcptr->volt[j+1]) != 2)
      /* If at end of file already, return error */
      return(FILE_END);

   /* Get datalogger moisture max and min */
   if(fgets(line_from_file, 60, qc_fp) == NULL)
      /* If at end of file already, return error */
      return(FILE_END);
   if(sscanf(line_from_file, " %f, %f ",&qcptr->mois[j],&qcptr->mois[j+1]) != 2)
      /* If at end of file already, return error */
      return(FILE_END);

   /* Get maximum allowable daily temp. change */
   if(fgets(line_from_file, 60, qc_fp) == NULL)
      /* If at end of file already, return error */
      return(FILE_END);
   if(sscanf(line_from_file, " %f ",&qcptr->maxtempchng) != 1)
      /* If at end of file already, return error */
      return(FILE_END);

   return(OK);
}  /* end ScanQC */
```

WQC.C

```
/* Include files */
#include <conio.h>      /* for console functions */
#include <stdio.h>      /* for file functions */
#include <process.h>    /* for spawn and exit functions */
#include <graph.h>      /* for clearscreen function */
#include <string.h>     /* for string manipulation functions */
#include "whdr.h"       /* defines, structs, functions for this program */
#include "wglobal.h"    /* global variables */


/*** QC and data formatting functions ***/


void RunQC(void)
/*-------------------------------------------------------------------------*/
/* Gets any unretrieved weather data from station (stored in ASCII file),  */
/*  reads one data set at a time, stores each in backup file, runs qc      */
/*  procedures on hourly and daily data sets, stores daily summaries in    */
/*  IBSNAT format in the current file, then deletes the raw data file.     */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns:  none.                                                         */
/* Calls:  CallStation                                                     */
/*         FormatData                                                      */
/*         ConvertRawData                                                  */
/*         DayQC                                                           */
/*         GetDay                                                          */
/*         GetHour                                                         */
/*         GetPrecip                                                       */
/*         GetQCValues                                                     */
/*         HourQC                                                          */
/*         OpenFile                                                        */
/*         OpenCurrentFile                                                 */
/*         OpenPreviousFile                                                */
/*         PrintArrayID                                                    */
/*         PrintDay                                                        */
/*         PrintFormattedDay                                               */
/*         PrintHour                                                       */
/*         PrintPrecip                                                     */
/*         SavePrevious                                                    */
/*         ScanArrayID                                                     */
/*-------------------------------------------------------------------------*/
{
struct day day_data[DAYS_PREVIOUS];  /* present and previous daily summaries */
struct hour hour_data[HOURS_PREVIOUS]; /* present, previous hourly summaries */
struct formatted_day formatted_data; /* daily summary in IBSNAT format */
struct qc qcvalues;  /* values for QC tests */
int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION];
                                        /* array of QC errors and actions */
int precip_time;  /* time of a precipitation occurrance */
float precip_amount;  /* amount of a precipitation occurrance */
int arrayid;    /* ID for hourly, daily, or precip occurrance summary */
FILE *rawdata_fp;     /* pointer to raw data file */
FILE *previous_fp;    /* pointer to previous data file */
FILE *backup_fp;      /* pointer to backup data file */
FILE *current_fp;     /* pointer to file of current IBSNAT-formatted data */
int open_result;      /* signifies if error occured during file opening */
int choice;           /* user's choice */
void CallStation(void);
void ConvertRawData(struct day *);
int DayQC(struct day *, struct day *, struct qc *,
      int [VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION], FILE *);
void GetDay(FILE *, struct day *);
void GetHour(FILE *, struct hour *);
void GetPrecip(FILE *, int *, float *);
void HourQC(struct hour *, struct hour *, struct hour *, struct qc *);
FILE *OpenCurrentFile(void);
FILE *OpenPreviousFile(struct day *, struct hour *, struct hour *);
void PrintArrayID(FILE *, int);
```

```
void PrintPrecip(FILE *, int, float);
void SavePrevious(FILE *, struct day *, struct hour *, struct hour *);
int ScanArrayID(FILE *);

   /* Ask if want to proceed */
   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\n\tAre you sure you want to call the weather station? ");
      choice = getche();
      if((choice == 'y') || (choice == 'Y')){
         /* Call weather station; data backup in raw data file */
/*        CallStation();*/
         break;}
      if((choice == 'n') || (choice == 'N'))
         return;}

   /* Tell user what is happening */
   _clearscreen(_GCLEARSCREEN);
   printf("\n\n\tNow performing QC on data and ");
   printf("adding daily data to IBSNAT file...\n");
   /* Open raw data file to begin reading in data */
   rawdata_fp = OpenFile(&open_result, info.raw_file, READ);
   /* If file does not exist, say so and quit */
   if(open_result == ERROR){
      printf("\n\n Could not open weather station raw data file!\n");
      printf("\t\t(press any key)");
      getch();
      return;}
   /* Load file of QC data into QC record; if no data, say so and quit */
   if(GetQCValues(&qcvalues) != OK){
      printf("\nInsufficient data in qc file!\n");
      printf("Use the 'View/Edit File of QC Values' function to\n");
      printf("   enter QC values.");
      fcloseall();
      printf("\t\t(press any key)");
      getch();
      return;}
   /* Open backup data file to append new data */
   backup_fp = OpenFile(&open_result, info.backup_file, APPEND);
   /* If couldn't open file, say so and quit */
   if(open_result == ERROR){
      printf("\n\n Could not open backup data file!\n");
      printf("\t\t(press any key)");
      getch();
      fcloseall();
      return;}
   /* Open file with previous hours'/day's data and put in array */
   previous_fp = OpenPreviousFile(&day_data[MINUS1], &hour_data[MINUS1],
         &hour_data[MINUS2]);
   /* Open current IBSNAT file to append new data */
   current_fp = OpenCurrentFile();
   /* Read array ID number from raw data file, then do appropriate case */
   while((arrayid = ScanArrayID(rawdata_fp)) != FILE_END){
      /* Print the array ID to the backup file */
      PrintArrayID(backup_fp, arrayid);
      switch(arrayid){
         /* If ID is for an hourly data set ... */
         case HOUR_ID:
            /* Read in line of hourly data */
            GetHour(rawdata_fp, &hour_data[PRESENT]);
            /* Put hourly data in backup file */
            PrintHour(backup_fp, &hour_data[PRESENT]);
            /* Do qc on hourly data */
            HourQC(&hour_data[PRESENT], &hour_data[MINUS1], &hour_data[MINUS2],
                  &qcvalues);
            /* Update 'previous' data */
            hour_data[MINUS2] = hour_data[MINUS1];
            hour_data[MINUS1] = hour_data[PRESENT];
            break;

         /* If ID is for a daily data set ... */
         case DAY_ID:
            /* Read in line of daily data */
```

```
            GetDay(rawdata_fp, &day_data[PRESENT]);
            /* Put daily data in backup file */
            PrintDay(backup_fp, &day_data[PRESENT]);
            /* Perform any data conversions on the raw data */
            ConvertRawData(&day_data[PRESENT]);
            /* Do qc on daily data */
            if(DayQC(&day_data[PRESENT], &day_data[MINUS1], &qcvalues,
                  erroractions, current_fp) == ERROR)
               break;
            /* Convert values to units needed for model */
            FormatData(&formatted_data, &day_data[PRESENT]);
            /* Put converted data into current file */
            PrintFormattedDay(current_fp, &formatted_data, erroractions);
            /* Make present data the previous day's data */
            day_data[MINUS1] = day_data[PRESENT];
            break;

         /* If ID is for a precip data set ... */
         case PREC_ID:
            /* Read in line of precip data */
            GetPrecip(rawdata_fp, &precip_time, &precip_amount);
            /* Put precip data in backup file */
            PrintPrecip(backup_fp, precip_time, precip_amount);
            break;}}   /* end switch, while */

   /* After end of raw data file is reached... */
   /* save the 'previous' data in a file for future qc */
   SavePrevious(previous_fp, &day_data[MINUS1], &hour_data[MINUS1],
         &hour_data[MINUS2]);
   /* close all files */
   fcloseall();
   /* Delete raw data file, so this data is not read again */
   remove(info.raw_file);

   return;
} /* end RunQC */


void AddDayToMissingDate(int *missingday_jdate, int *missingday_year,
      int *presentday_year)
/*-------------------------------------------------------------------------*/
/* Adds one day to the previous day's date to get the date for the day     */
/*  which was missing and is being filled with other data; if the date     */
/*  begins a new year, the program info and year of present day being run  */
/*  through QC are incremented, if necessary.                              */
/* Receives: *missingday_jdate:  address of the previous day's date; value */
/*                               is changed to become missing day's date   */
/*           *missingday_year:  address of the previous day's year; value  */
/*                              is changed to become missing day's year    */
/*           *presentday_year:  address of the year of the present data    */
/*                              set being run through QC                   */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: SaveProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
{
   /* If the previous day's year is a leap year... */
   if((*missingday_year % 4) == 0)
      /* If the previous day's date is the last day of the year... */
      if(*missingday_jdate == 366){
         /* Set the missing day's date and year accordingly */
         *missingday_jdate = 1;
         *missingday_year = *missingday_year + 1;
         /* If the present day's year has not been incremented, do so */
         if(*presentday_year != *missingday_year)
            *presentday_year = *missingday_year;
         /* If the program info year has not been incremented, do so */
         if(info.year != *missingday_year){
            info.year++;
            SaveProgramInfo();}
         return;}
      /* If the previous day's date is not the last day of the year... */
```

```
      else{
         /* Just add one to the date */
         *missingday_jdate = *missingday_jdate + 1;
         return;}

   /* If the previous day's year is not a leap year... */
   /* If the previous day's date is the last day of the year ... */
   if(*missingday_jdate == 365){
      /* Set the missing day's date accordingly */
      *missingday_jdate = 1;
      *missingday_year = *missingday_year + 1;
      /* If the present day's year has not been incremented, do so */
      if(*presentday_year != *missingday_year)
         *presentday_year = *missingday_year;
      /* If the program info year has not been incremented, do so */
      if(info.year != *missingday_year){
         info.year++;
         SaveProgramInfo();}
      return;}
   /* If the previous day's date is not the last day of the year... */
   else{
      /* Just add one to the date */
      *missingday_jdate = *missingday_jdate + 1;
      return;}
} /* end AddDayToMissingDate */


void AveragePresentAndPreviousData(struct day *missingday_ptr,
      struct day *presentday_ptr, struct day *previousday_ptr)
/*-------------------------------------------------------------------------*/
/* Averages the max/min temp, SR, and precip of the day before missing     */
/*  data begins and of the present day's data set being run through QC;    */
/*  this is done when the user wants to make missing data like the average */
/*  of these two days' data                                                */
/* Receives: *missingday_ptr:  address of missing daily summary data       */
/*           *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* Put the averages in the missing day's data */
   missingday_ptr->sr = (presentday_ptr->sr + previousday_ptr->sr)/2.0;
   missingday_ptr->maxt = (presentday_ptr->maxt + previousday_ptr->maxt)/2.0;
   missingday_ptr->mint = (presentday_ptr->mint + previousday_ptr->mint)/2.0;
   missingday_ptr->prec = (presentday_ptr->prec + previousday_ptr->prec)/2.0;
} /* end AveragePresentAndPreviousData */


void CallStation()
/*-------------------------------------------------------------------------*/
/* Calls the Campbell 'telcom' program which retrieves data from the       */
/*  weather station datalogger and stores it in a comma delineated ASCII   */
/*  file.                                                                  */
/* Receives: none.                                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* Tell user what is happening */
   _clearscreen(_GCLEARSCREEN);
   printf("\n\n\tNow retrieving weather data from station...\n\n");
   printf("\t(Please wait.)\n\n");
   /* Call the telcom program (used to retrieve data from station) */
   spawnl(P_WAIT, "telcom", "telcom", info.sta_name, "/c", NULL);
   return;
} /* end CallStation */
```

```
int ChangeDateOrDeleteData(void)
/*-------------------------------------------------------------------------*/
/* Presents the user with the options to change the date of the data set of*/
/*  the present day being run through QC or to delete the data set (not    */
/*  save the data in the current file) and then returns the number of the  */
/*  user's choice.                                                         */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: 1:  if the user wants to change the date                       */
/*          2:  if the user wants to delete the data                       */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int ascii_number; /* ASCII value of the user's choice */
int choice;  /* user's choice */

   /* Print choices */
   printf("\nChoose action to take:\n");
   printf("\t1.  Change the date of the present day's dataset\n");
   printf("\t2.  Delete present day's dataset\n");
   printf("Your choice:  ");
   /* Repeat question until get a valid answer */
   while(TRUE){
      /* Get the user's answer */
      ascii_number = getche();
      /* Convert ASCII value of answer to the correct number */
      choice = ascii_number - 48;
      /* Return choice if a valid answer */
      switch(choice){
         /* Change date */
         case 1:
         case 2:
            return(choice);
            break;
         /* If not a valid choice, ask again */
         default:
            printf("\nYour choice:  ");}}
}  /* end ChangeDateOrDeleteData */


int ChangeDateOrFillMissingData(struct day *previousday_ptr)
/*-------------------------------------------------------------------------*/
/* Presents the user with the options to change the date of the data set of*/
/*  the present day being run through QC or to fill-in data for the days   */
/*  which are missing, and then returns the number of the user's choice.   */
/* Receives: *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/* Globals: none.                                                          */
/* Returns: 1: if the user wants to change the date                        */
/*          2: if the user wants to fill-in the missing data               */
/* Calls: PrintDaysDataToScreen                                            */
/*-------------------------------------------------------------------------*/
{
int ascii_number; /* ASCII value of the user's choice */
int choice;  /* user's choice */

   /* Print a summary of the previous day's data */
   printf("\tSummary of this day's data:\n");
   PrintDaysDataToScreen(previousday_ptr);
   /* Print options */
   printf("\nChoose action to take:\n");
   printf("\t1.  Change the date of the present day's dataset\n");
   printf("\t2.  Fill in data between these two dates\n");
   printf("Your choice:  ");
   /* Repeat question until get a valid answer */
   while(TRUE){
      /* Get answer from user */
      ascii_number = getche();
      /* Convert ASCII value of answer to the correct number */
      choice = ascii_number - 48;
      /* Return choice if a valid answer */
      switch(choice){
```

```
         /* Change date */
         case 1:
         case 2:
            return(choice);
            break;
         /* If not a valid choice, ask again */
         default:
            printf("\nYour choice:  ");}}
} /* end ChangeDateOrFillMissingData */


int ChooseHowToFillMissingData(struct day *presentday_ptr,
      struct day *previousday_ptr)
/*-------------------------------------------------------------------------*/
/* Presents the user with the options of how to fill-in the data for the   */
/*  missing day(s):  like the data of the day before the missing day(s)    */
/*  begin; like the data of the present day being run through QC; or like  */
/*  an average of these two days; then returns the number of the user's    */
/*  choice.                                                                */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/* Globals: month_name                                                     */
/* Returns: 1:  if the user wants to make the data like the previous day's */
/*          2:  if the user wants to make the data like the present day's  */
/*          3:  if the user wants to make the data like an ave. of the two */
/* Calls: CalDate                                                          */
/*        PrintDaysDataToScreen                                            */
/*-------------------------------------------------------------------------*/
{
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int previousday_month; /* month of the previous daily summary */
int previousday_day; /* day of the present daily summary */
int ascii_number; /* ASCII value of the user's choice */
int choice;  /* user's choice */

   /* Change Julian date to calendar date for showing any errant data */
   CalDate(presentday_ptr->jdate, presentday_ptr->year, &presentday_month,
         &presentday_day);
   CalDate(previousday_ptr->jdate, previousday_ptr->year, &previousday_month,
         &previousday_day);

   /* Print message to user */
   printf("\n\nYou have chosen to fill missing data.  You must choose one of");
   printf("\n   three options for filling the data.  If you wish to enter");
   printf("\n   individual values for each missing day, you may do so in the");
   printf("\n   'Edit Current Data File' section of this program\n\n");
   printf("\tSummary of present day's data (%s %d, %d):\n",
         month_name[presentday_month], presentday_day, presentday_ptr->year);
   PrintDaysDataToScreen(presentday_ptr);
   printf("\tSummary of data for last day in current file (%s %d, %d):\n",
         month_name[previousday_month], previousday_day, previousday_ptr->year);
   PrintDaysDataToScreen(previousday_ptr);
   /* Print options */
   printf("\nChoose option for fill:\n");
   printf("\t1.  Make data equal to data of last day in current file\n");
   printf("\t2.  Make data equal to present day's data\n");
   printf("\t3.  Make data equal to average of these two days' data\n");
   printf("Your choice:  ");
   /* Repeat question until get a valid answer */
   while(TRUE){
      /* Get answer from user */
      ascii_number = getche();
      /* Convert ASCII value of answer to the correct number */
      choice = ascii_number - 48;
      /* Return choice if a valid answer */
      switch(choice){
         /* Chosen an option */
         case 1:
         case 2:
```

```
         case 3:
            return(choice);
            break;
         /* If not a valid choice, ask again */
         default:
            printf("\nYour choice:  ");}}
} /* end ChooseHowToFillMissingData */


void ConvertRawData(struct day *presentday_ptr)
/*-------------------------------------------------------------------------*/
/* Converts ave. solar rad. reading to total daily solar rad. in MJ/m2,    */
/*  and subtracts one day from the Julian date.                            */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* Make ave. SR reading from station into total daily SR */
   presentday_ptr->sr = presentday_ptr->sr * CONVERTAVESR;

   /* Subtract one from Julian day because data is actually for previous day */
   /* Make adjustment for first day of year (set equal to last day of year */
   if(presentday_ptr->jdate == 1){
      if((info.year % 4) == 0)
         presentday_ptr->jdate = 366;
      else
         presentday_ptr->jdate = 365;}
   else
      presentday_ptr->jdate = presentday_ptr->jdate - 1;
} /* end ConvertRawData */


int DayQC(struct day *presentday_ptr, struct day *previousday_ptr, struct qc
      *qc_ptr, int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION],
      FILE *current_fp)
/*-------------------------------------------------------------------------*/
/* Checks a set of daily data to see if it is reasonable                   */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           *qc_ptr:  address of the QC values                            */
/*           erroractions:  array of QC errors and actions                 */
/*           *current_fp: pointer to current data file                     */
/* Globals: none.                                                          */
/* Returns:  PASS: for all but instance listed below                       */
/*           FAIL: if the Julian day failed a qc test and the user does    */
/*                 not want the data to be put in IBSNAT file              */
/* Calls: CalDate                                                          */
/*        DayJulianDateQC                                                  */
/*        DayMoisQC                                                        */
/*        DayPrecQC                                                        */
/*        DayRHQC                                                          */
/*        DaySRQC                                                          */
/*        DayTempQC                                                        */
/*        DayVoltQC                                                        */
/*        DayWindQC                                                        */
/*        FigureProperYear                                                 */
/*        FillMissingData                                                  */
/*        FixError                                                         */
/*-------------------------------------------------------------------------*/
{
int warning; /* number indicating type of QC error found (if any) */
float temporary_value;  /* used to hold a value which failed QC so user may */
                        /*    change it */
float dummy_float = 0.0;  /* used in a function call when not all parameters */
                          /*    are needed */
int i; /* loop counter */
int j; /* loop counter */
```

```
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int difference; /* difference between the present day's Julian date and the */
                /*    previous day's Julian date */
int jdate_error = 0;  /* error code (if any) resulting from Julian date QC */
int jdate_action = 0; /* action code (if any) resulting from Julian date QC */
int DayJulianDateQC(struct day *, struct day *, int *, int *, int *);
int DayMoisQC(float, struct qc *);
int DayRHQC(float, struct qc *, float, float, int);
int DayVoltQC(float, struct qc *);
int DayWindQC(float, float);
int FigureProperYear(int);
void FillMissingData(struct day *, struct day *, int, FILE *);
int FixError(float *, float, int, int, struct qc *, float);

   /* Set array of errors/actions to zero */
   for(i=0; i<=VALUES_WITH_ERRORS-1; i++)
      for(j=0; j<=ONE_ERROR_ONE_ACTION-1; j++)
         erroractions[i][j] = 0;

   /* If no previous day's data, signify */
   if(previousday_ptr->jdate == NO_DATA){
      previousday_ptr->maxt = (float) NO_DATA;
      previousday_ptr->mint = (float) NO_DATA;
      previousday_ptr->maxrh = (float) NO_DATA;
      previousday_ptr->minrh = (float) NO_DATA;
      previousday_ptr->maxws = (float) NO_DATA;}

   /* Determine the proper year for the day's data */
   presentday_ptr->year = FigureProperYear(previousday_ptr->jdate);

   /* Test the Julian date */
   if(DayJulianDateQC(presentday_ptr, previousday_ptr, &jdate_error,
            &jdate_action, &difference) != PASS)
      /* User does not want to put the errant day's data in current file */
      return(FAIL);
   erroractions[JDATE][QC_ERROR] = jdate_error;
   erroractions[JDATE][QC_ACTION] = jdate_action;

   /* Change Julian date to calendar date for showing any errant data */
   CalDate(presentday_ptr->jdate, presentday_ptr->year, &presentday_month,
         &presentday_day);

   /* Test max temp */
   if((warning = DayTempQC(presentday_ptr->maxt, presentday_month, qc_ptr,
         previousday_ptr->maxt, presentday_ptr->mint, VALDMXT)) != PASS){
      /* Print the warning if max temp fails any one qc test */
      PrintWarning(warning, presentday_month, presentday_day,
            presentday_ptr->year, presentday_ptr->hour, presentday_ptr->maxt,
            VALDMXT, FRNHT);
      if(warning == MAXTOUTRNG)
         printf("   Extreme values:     %6.2f     %6.2f\n",
               qc_ptr->temp[presentday_month][0],
               qc_ptr->temp[presentday_month][1]);
      /* Put the error code in the error/action array */
      erroractions[MAXTEMP][QC_ERROR] = warning;
      /* Put the max temp into a temporary variable */
      temporary_value = presentday_ptr->maxt;
      /* Put the action the user takes into the error/action array and */
      /*    allow the max temp to be changed */
      erroractions[MAXTEMP][QC_ACTION] = FixError(&temporary_value,
            previousday_ptr->maxt, VALDMXT, presentday_month, qc_ptr,
            presentday_ptr->mint);
      presentday_ptr->maxt = temporary_value;
      /* If the problem was max temp < min temp, allow for changing of */
      /*    min temp also */
      if(warning == MAXTLESSMINT){
         erroractions[MINTEMP][QC_ERROR] = warning;
         temporary_value = presentday_ptr->mint;
         erroractions[MINTEMP][QC_ACTION] = FixError(&temporary_value,
               previousday_ptr->maxt, VALDMNT, presentday_month, qc_ptr,
               dummy_float);
         presentday_ptr->mint = temporary_value;}}
```

```
/* Test min temp */
if((warning = DayTempQC(presentday_ptr->mint, presentday_month, qc_ptr,
      previousday_ptr->mint, dummy_float, VALDMNT)) != PASS){
   PrintWarning(warning, presentday_month, presentday_day,
         presentday_ptr->year, presentday_ptr->hour, presentday_ptr->mint,
         VALDMNT, FRNHT);
   if(warning == MINTOUTRNG)
      printf("   Extreme values:     %6.2f     %6.2f\n",
            qc_ptr->temp[presentday_month][0],
            qc_ptr->temp[presentday_month][1]);
   erroractions[MINTEMP][QC_ERROR] = warning;
   temporary_value = presentday_ptr->mint;
   erroractions[MINTEMP][QC_ACTION] = FixError(&temporary_value,
         previousday_ptr->mint, VALDMNT, presentday_month, qc_ptr,
         dummy_float);
   presentday_ptr->mint = temporary_value;}

/* Test solar rad */
if((warning = DaySRQC(presentday_ptr->sr, presentday_month, qc_ptr)) !=
      PASS){
   PrintWarning(warning, presentday_month, presentday_day,
         presentday_ptr->year, presentday_ptr->hour, presentday_ptr->sr,
         VALDTSR, MJPDA);
   if(warning == SOLAROUTRNG)
      printf("   Extreme values:     %6.2f     %6.2f\n",
            qc_ptr->sr[presentday_month][0],
            qc_ptr->sr[presentday_month][1]);
   erroractions[SOLAR][QC_ERROR] = warning;
   temporary_value = presentday_ptr->sr;
   erroractions[SOLAR][QC_ACTION] = FixError(&temporary_value,
         previousday_ptr->sr, VALDTSR, presentday_month, qc_ptr,
         dummy_float);
   presentday_ptr->sr = temporary_value;}

/* Test precip */
if((warning = DayPrecQC(presentday_ptr->prec, presentday_month, qc_ptr)) !=
      PASS){
   PrintWarning(warning, presentday_month, presentday_day,
         presentday_ptr->year, presentday_ptr->hour, presentday_ptr->prec,
         VALDTPR, INCHS);
   if(warning == PRECIPOUTRNG)
      printf("   Extreme values:     %6.2f     %6.2f\n",
            qc_ptr->prec[presentday_month][0],
            qc_ptr->prec[presentday_month][1]);
   erroractions[PRECIP][QC_ERROR] = warning;
   temporary_value = presentday_ptr->prec;
   erroractions[PRECIP][QC_ACTION] = FixError(&temporary_value,
         previousday_ptr->prec, VALDTPR, presentday_month, qc_ptr,
         dummy_float);
   presentday_ptr->prec = temporary_value;}

/* Because the rest of the values are not used in the IBSNAT file, */
/*  just print a warning if any one fails a test /*
/* Test min rel. humidity */
if((warning = DayRHQC(presentday_ptr->maxrh, qc_ptr, previousday_ptr->maxrh,
      presentday_ptr->minrh, VALDMXR)) != PASS){
   PrintWarning(warning, presentday_month, presentday_day,
         presentday_ptr->year, presentday_ptr->hour, presentday_ptr->maxrh,
         VALDMXR, PRCNT);
   printf("\t\t(press any key)");
   getch();}
/* Test max rel. humidity */
if((warning = DayRHQC(presentday_ptr->minrh, qc_ptr, previousday_ptr->minrh,
      dummy_float, VALDMNR)) != PASS){
   PrintWarning(warning, presentday_month, presentday_day,
         presentday_ptr->year, presentday_ptr->hour, presentday_ptr->minrh,
         VALDMNR, PRCNT);
   printf("\t\t(press any key)");
   getch();}
/* Test max wind speed */
if((warning = DayWindQC(presentday_ptr->maxws, previousday_ptr->maxws))
      != PASS){
   PrintWarning(warning, presentday_month, presentday_day,
```

```
            presentday_ptr->year, presentday_ptr->hour, presentday_ptr->maxws,
            VALDMWS, MLPHR);
         printf("\t\t(press any key)");
         getch();}
      /* Test datalogger voltage */
      if((warning = DayVoltQC(presentday_ptr->volt, qc_ptr)) != PASS){
         PrintWarning(warning, presentday_month, presentday_day,
            presentday_ptr->year, presentday_ptr->hour, presentday_ptr->volt,
            VALDVLT, VOLTS);
         printf("\t\t(press any key)");
         getch();}
      /* Test datalogger moisture */
      if((warning = DayMoisQC(presentday_ptr->mois, qc_ptr)) != PASS){
         PrintWarning(warning, presentday_month, presentday_day,
            presentday_ptr->year, presentday_ptr->hour, presentday_ptr->mois,
            VALDMOI, INDEX);
         printf("\t\t(press any key)");
         getch();}

      /* After checking present day's data, if need to fill in missing data, */
      /*    do so */
      if(difference > 1)
         FillMissingData(presentday_ptr, previousday_ptr,
            difference, current_fp);

      /* Return passing value */
      return(PASS);
   }  /* end DayQC */


int DayJulianDateQC(struct day *presentday_ptr, struct day *previousday_ptr,
      int *jdate_error, int *jdate_action, int *difference)
/*-------------------------------------------------------------------------*/
/* Checks the Julian date to see if it is within the range of 1-366 and    */
/*    if it follows the previous day's date.                               */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           *jdate_error: address of the QC error from Julian date tests  */
/*           *jdate_action: address of QC action from Julian date tests    */
/*           *difference: address of the difference between the present    */
/*                        and previous Julian dates                        */
/* Globals: info                                                           */
/* Returns: PASS for all but one instance                                  */
/*          FAIL only if the Julian day failed a qc test and the user does */
/*          not want the data to be put in IBSNAT file                     */
/* Calls: CalDate                                                          */
/*        ChangeDateOrDeleteData                                           */
/*        ChangeDateOrFillMissingData                                      */
/*        DecideOnFirstDaysData                                            */
/*        GetDifference                                                    */
/*        GetNewDateFromUser                                               */
/*        PrintJulianDateWarning                                           */
/*-------------------------------------------------------------------------*/
{
int choice;   /* user's choice */
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int previousday_month; /* month of the previous daily summary */
int previousday_day; /* day of the present daily summary */
int warning = 0;  /* number indicating type of QC error found (if any) */
int first_time_through = TRUE; /* idicates the first time through Julian */
                               /*    QC tests */
int DecideOnFirstDaysData(struct day *);
void PrintJulianDateWarning(int, struct day *, int, int, int, int, int);
int ChangeDateOrDeleteData(void);
int ChangeDateOrFillMissingData(struct day *);
int GetNewDateFromUser(int, int *);
int GetDifference(int, int, int);

   /* Get month and day of previous day's data, for showing to user */
```

```
if(previousday_ptr->jdate != NO_DATA)
   CalDate(previousday_ptr->jdate, previousday_ptr->year,
         &previousday_month, & previousday_day);

/* Repeat until one of the following conditions allows returning */
while(TRUE){
   /* If no previous day's data ... */
   if(previousday_ptr->jdate == NO_DATA)
      /* Let user decide to keep or delete first day's data */
      return(DecideOnFirstDaysData(presentday_ptr));

   /* For all data after first day of data is collected ... */
   /* Get month and day of present day's data, and */
   /*    check to make sure day number is in proper range */
   if(CalDate(presentday_ptr->jdate, presentday_ptr->year, &presentday_month,
         &presentday_day) == CD_INVALID_JDATE){
      /* Date is out of range */
      warning = DATEOUTRNG;
      PrintJulianDateWarning(warning, presentday_ptr, presentday_month,
            presentday_day, previousday_month, previousday_day,
            previousday_ptr->year);
      /* Get the user's choice of what to do */
      choice = ChangeDateOrDeleteData();
      switch(choice){
         /* User wants to change the date */
         case 1:
            /* Get the new date */
            presentday_ptr->jdate = GetNewDateFromUser(previousday_ptr->year,
                  &presentday_ptr->year);
            /* Record the first error encountered */
            if(first_time_through){
               *jdate_error = warning;
               *jdate_action = choice;
               first_time_through = FALSE;}
            break;
         /* User wants to delete the data */
         case 2:
            return(FAIL);
            break;}
      /* Check the new date the user just entered */
      continue;}

   /* Get difference of present date - previous date */
   /*  Make allowance for end of year */
   *difference = GetDifference(presentday_ptr->jdate, previousday_ptr->jdate,
         previousday_ptr->year);

   /* If present day follows previous day by one day, return PASS */
   if(*difference == 1)
      return(PASS);

   /* If difference is 0 or negative, have an errant date */
   if(*difference < 1){
      /* Date is out of order */
      warning = DATEOUTORDR;
      PrintJulianDateWarning(warning, presentday_ptr, presentday_month,
            presentday_day, previousday_month, previousday_day,
            previousday_ptr->year);
      /* Get the user's choice of what to do */
      choice = ChangeDateOrDeleteData();
      switch(choice){
         /* User wants to change the date */
         case 1:
            /* Get new date from user */
            presentday_ptr->jdate = GetNewDateFromUser(previousday_ptr->year,
                  &presentday_ptr->year);
            /* Record first error encountered */
            if(first_time_through){
               *jdate_error = warning;
               *jdate_action = choice;
               first_time_through = FALSE;}
            break;
         /* User wants to delete the data */
```

```
            case 2:
               return(FAIL);
               break;}
         /* Check the new date the user just entered */
         continue;}

      /* If difference is greater than 1, have an errant date or have */
      /*    one or more daily summaries missing */
      if(*difference > 1){
         /* Some days of data are missing (or date is wrong) */
         warning = DATEMISSING;
         PrintJulianDateWarning(warning, presentday_ptr, presentday_month,
               presentday_day, previousday_month, previousday_day,
               previousday_ptr->year);
         /* Get user's choice of what to do */
         choice = ChangeDateOrFillMissingData(previousday_ptr);
         switch(choice){
            /* User wants to change the date */
            case 1:
               presentday_ptr->jdate = GetNewDateFromUser(previousday_ptr->year,
                     &presentday_ptr->year);
               /* Record the first error encountered */
               if(first_time_through){
                  *jdate_error = warning;
                  *jdate_action = choice;
                  first_time_through = FALSE;}
               break;
            /* User wants to fill-in the missing data */
            case 2:
               /* Give user message that data will be filled later */
               /* That data will be filled is indicated by difference value */
               printf("\n\nAfter this present data is run through the ");
               printf("remaining QC procedures,\n   you will be allowed to ");
               printf("choose how the missing data should be filled\n");
               printf("(press any key)");
               getch();
               /* If first time through, nothing is wrong with this date, */
               /*    just need to fill in the missing data */
               if(first_time_through){
                  *jdate_error = 0;
                  *jdate_action = 0;
                  first_time_through = FALSE;}
               return(PASS);
               break;}
         /* Check new date just entered by user */
         continue;}}
} /* end DayJulianDateQC */


int DayMoisQC(float checkvalue, struct qc *qc_ptr)
/*-------------------------------------------------------------------------*/
/* Checks the datalogger moisture value.                                   */
/* Receives: checkvalue:  value to check                                   */
/*           *qc_ptr: address of QC values                                 */
/* Globals: none.                                                          */
/* Returns: PASS if value passes all tests                                 */
/*          FAIL if value fails any one test                               */
/* Calls: CheckExtreme                                                     */
/*-------------------------------------------------------------------------*/
{
   /* Make sure moisture is within range */
   if(CheckExtreme(checkvalue, qc_ptr->mois) != PASS)
      return(MOISTROUTRNG);
   /* If no check failures, return passing value */
} /* end DayMoisQC */


int DayRHQC(float checkvalue, struct qc *qc_ptr, float previous1value,
            float opposite_value, int val_code)
/*-------------------------------------------------------------------------*/
/* Checks relative humidity values.                                        */
/* Receives: checkvalue:  value to check                                   */
/*           *qc_ptr: address of QC values                                 */
```

```
/*          previous1value:  immediately previous value for comparison    */
/*          opposite_value:  opposite value for comparison                */
/*          val_code:  number signifying what the value represents        */
/* Globals: none.                                                         */
/* Returns: PASS if value passes all tests                                */
/*          FAIL if value fails any one test                              */
/* Calls: CheckConstant                                                   */
/*        CheckExtreme                                                    */
/*        CheckMaxMin                                                     */
/*------------------------------------------------------------------------*/
{
   /* Make sure RH is within range */
   if(CheckExtreme(checkvalue, qc_ptr->rh) != PASS)
      return(RHOUTRNG);
   /* Make sure RH is not exactly the same as yesterday's */
   if(CheckConstant(checkvalue, previous1value, (float) NO_DATA) != PASS)
      return(RHCONSTNT);
   /* Make sure max RH is not less than min RH */
   /* Only run this check on the max value */
   if(val_code == VALDMXR)
      if(CheckMaxMin(checkvalue, opposite_value) != PASS)
         return(MXRHLESSMNRH);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end DayRHQC */


int DayWindQC(float checkvalue, float previous1value)
/*------------------------------------------------------------------------*/
/* Checks the wind speed value.                                           */
/* Receives: checkvalue:  value to check                                  */
/*           previous1value: immediately previous value for comparison    */
/* Globals: none.                                                         */
/* Returns: PASS if value passes all tests                                */
/*          FAIL if value fails any one test                              */
/* Calls: CheckConstant                                                   */
/*------------------------------------------------------------------------*/
{
   /* Make sure wind is not exactly the same as yesterday's */
   if(CheckConstant(checkvalue, previous1value, (float) NO_DATA) != PASS)
      return(WINDCONSTNT);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end DayWindQC */


int DayVoltQC(float checkvalue, struct qc *qc_ptr)
/*------------------------------------------------------------------------*/
/* Checks the datalogger battery voltage value.                           */
/* Receives: checkvalue:  value to check                                  */
/*           *qc_ptr: address of QC values                                */
/* Globals: none.                                                         */
/* Returns: PASS if value passes all tests                                */
/*          FAIL if value fails any one test                              */
/* Calls: CheckExtreme                                                    */
/*------------------------------------------------------------------------*/
{
   /* Make sure voltage is within range */
   if(CheckExtreme(checkvalue, qc_ptr->volt) != PASS)
      return(BATTRYOUTRNG);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end DayVoltQC */


int DecideOnFirstDaysData(struct day *presentday_ptr)
/*------------------------------------------------------------------------*/
/* Lets user decide what to do with the first daily summary collected.    */
/* Receives: *presentday_ptr:  address of the daily summary data for the  */
/*                             present day                                */
/* Globals: info                                                          */
/*          month_name                                                    */
/* Returns: PASS if user wants to keep data                               */
```

```
/*          FAIL if user wants to delete data                              */
/* Calls: CalDate                                                         */
/*        PrintDaysDataToScreen                                           */
/*        SaveProgramInfo                                                 */
/*        GetNewFirstDateFromUser                                         */
/*------------------------------------------------------------------------*/
{
int warning; /* number indicating type of QC error found (if any) */
int choice; /* user's choice */
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int GetNewFirstDateFromUser(int *);

   /* Loop until either keep or delete this day's data */
   while(TRUE){
      /* Check to make sure day number is in proper range */
      if(CalDate(presentday_ptr->jdate, presentday_ptr->year, &presentday_month,
            &presentday_day) == CD_INVALID_JDATE)
         /* Date is out of range */
         warning = DATEOUTRNG;

      /* Print message */
      printf("\n\n\n");
      printf("\t--- Report from:  QC procedures on raw data ");
      printf("from weather station ---");
      printf("\n\nNOTICE:  This is the first daily summary collected\n");
      printf("   Recorded date:  ");
      if(warning == DATEOUTRNG)
         printf("[invalid date]\n");
      else
         printf("%s %d, %d\n", month_name[presentday_month], presentday_day,
               presentday_ptr->year);
      printf("\tSummary of this day's data:\n");
      PrintDaysDataToScreen(presentday_ptr);
      /* Print choices */
      printf("\nChoose action to take:\n");
      printf("\t1.  Keep data as is\n");
      printf("\t2.  Delete this data\n");
      printf("\t3.  Change the date for this data\n");
      printf("\t(Must choose 2. or 3. if date is invalid)");
      /* State question and get answer until a valid answer is given */
      while(TRUE){
         printf("\nYour choice:  ");
         choice = getche();
         switch(choice){
            /* If keep date the same, return PASS value */
            case '1':
               /* Don't allow this choice if date is invalid */
               if(warning != DATEOUTRNG)
                  return(PASS);
               break;
            /* If not saving data, return FAIL value */
            case '2':
               return(FAIL);
               break;
            /* If changing date, get new date; convert to Julian date */
            case '3':
               presentday_ptr->jdate =
                     GetNewFirstDateFromUser(&presentday_ptr->year);
               /* Make sure program info year is same as the entered year */
               if(info.year != presentday_ptr->year){
                  info.year = presentday_ptr->year;
                  SaveProgramInfo();}
               break;
            /* Have user enter a correct choice if did not enter one */
            default:
               printf("\n*** Enter proper choice ***\n");}
      /* Re-check new date and allow user to change it again or keep it */
      if(choice == '3')
         break;}}
} /* end DecideOnFirstDaysData */
```

```
int FigureProperYear(int previousday_jdate)
/*-------------------------------------------------------------------------*/
/* Figures the year for the present day's date                             */
/* Receives: previousday_jdate: Julian date of the previous day's data     */
/* Globals: info                                                           */
/* Returns: the year for the present day's date                            */
/* Calls: SaveProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
{
   /* If the current year is a leap year ... */
   if((info.year %4) == 0){
      /* Perform test on previous day because it has gone thru QC already */
      /* If the previous day was the last day of the current year ... */
      if(previousday_jdate == 366){
         /* Increment year and update program info */
         info.year++;
         SaveProgramInfo();}}

   /* If the current year is not a leap year ... */
   /* If the previous day was the last day of the current year ... */
   else if(previousday_jdate == 365){
      /* Increment year and update program info */
      info.year++;
      SaveProgramInfo();}
   /* Return the proper year */
   return(info.year);
}  /* end FigureProperYear */


void FillMissingData(struct day *presentday_ptr, struct day *previousday_ptr,
      int difference, FILE *current_fp)
/*-------------------------------------------------------------------------*/
/* Fills in data for the date(s) for which data are missing                */
/* Receives: *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           difference: difference between the present day's Julian date  */
/*                       and the previous day's Julian date                */
/*           *current_fp:  pointer to the current data file                */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: FormatData                                                       */
/*        PrintFormattedDay                                                */
/*        AddDayToMissingDate                                              */
/*        AveragePresentAndPreviousData                                    */
/*        ChooseHowToFillMissingData                                       */
/*-------------------------------------------------------------------------*/
{
int i; /* loop counter */
int j; /* loop counter */
int k; /* loop counter */
int choice; /* user's choice */
int missingday_jdate; /* Julian date of missing daily summary */
int missingday_year; /* year of missing daily summary */
struct day missingday; /* data of missing daily summary */
struct formatted_day formatted_data; /* daily summary in IBSNAT format */
int missing_erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION];
int ChooseHowToFillMissingData(struct day *, struct day *);
void AddDayToMissingDate(int *, int *, int *);
void AveragePresentAndPreviousData(struct day *, struct day *, struct day *);

   choice = ChooseHowToFillMissingData(presentday_ptr, previousday_ptr);

   /* To start with, set the missing data's date = previous day's date */
   missingday_jdate = previousday_ptr->jdate;
   missingday_year = previousday_ptr->year;

   /* Repeat for each missing day */
   for(k=1; k<=difference-1;k++){
      /* Perform proper action depending on user's choice */
      switch(choice){
```

```
            /* Make missing data equal to previous day's data */
            case 1:
               /* Set missing data = previous data */
               missingday = *previousday_ptr;
               break;
            /* Make missing data equal to present day's data */
            case 2:
               /* Set missing data = present data */
               missingday = *presentday_ptr;
               break;
            /* Make missing data equal to ave. of previous and present data */
            case 3:
               /* set missing data = previous to get non-averaged data stored */
               missingday = *previousday_ptr;
               /* Get averages */
               AveragePresentAndPreviousData(&missingday, presentday_ptr,
                     previousday_ptr);
               break;}

         /* Add one day to missing data's day */
         /* Check for year change */
         AddDayToMissingDate(&missingday_jdate, &missingday_year,
               &presentday_ptr->year);
         missingday.jdate = missingday_jdate;
         missingday.year = missingday_year;

         /* Set array of missing errors/actions to zero */
         for(i=0; i<=VALUES_WITH_ERRORS-1; i++)
            for(j=0; j<=ONE_ERROR_ONE_ACTION-1; j++)
               missing_erroractions[i][j] = 0;
         /* Put error code and choice in action array */
         missing_erroractions[JDATE][QC_ERROR] = FILLEDDATA;
         missing_erroractions[JDATE][QC_ACTION] = choice;
         /* Convert values to units needed for model */
         FormatData(&formatted_data, &missingday);
         /* Put converted data into current file */
         PrintFormattedDay(current_fp, &formatted_data, missing_erroractions);}
} /* end FillMissingData */


int FixError(float *value_to_fix, float previous1value, int val_code,
      int presentday_month, struct qc *qc_ptr, float opposite_value)
/*-------------------------------------------------------------------------*/
/* Lets user decide what to do with a value which has failed QC.           */
/* Receives: *value_to_fix:  address of value that failed QC               */
/*           previous1value:  immediately previous value for comparison    */
/*           val_code:  number signifying what the value represents        */
/*           presentday_month:  month of the present daily summary         */
/*           *qc_ptr: address of QC values                                 */
/*           opposite_value:  opposite value for comparison                */
/* Globals: none.                                                          */
/* Returns: VALUE_UNCHANGED:  if user wants to keep value unchanged        */
/*          VALUE_EQUALS_PREVIOUS:  if user wants value set equal to the   */
/*                                  previous day's value                   */
/*          NEW_VALUE_ENTERED:  if user entered a new value                */
/* Calls: QCNewValue                                                       */
/*-------------------------------------------------------------------------*/
{
float newvalue; /* new value entered by user for one which failed QC */
int ascii_number; /* ASCII value of user's choice */
int choice; /* user's choice */
int keeping_newvalue = FALSE; /* indicates if the user wants to keep or */
                              /* change new value which failed QC tests */

   printf("   Yesterday's value:  %6.2f\n\n", previous1value);
   printf("Choose action to take:\n");
   printf("   1.  Keep value the same\n");
   printf("   2.  Change value to yesterday's value\n");
   printf("   3.  Enter a replacement value\n");
   printf("Your choice:  ");
   /* Repeat question until get a valid answer */
   while(TRUE){
      ascii_number = getche();
```

```
      /* Convert ASCII value of answer to the correct number */
      choice = ascii_number - 48;
      /* Do proper action depending on choice */
      switch(choice){
         /* Keep value the same */
         case 1:
            return(VALUE_UNCHANGED);
            break;
         /* Set value equal to yesterday's value */
         case 2:
            *value_to_fix = previous1value;
            return(VALUE_EQUALS_PREVIOUS);
            break;
         /* Let user enter a replacement value */
         case 3:
            while(keeping_newvalue == FALSE){
               printf("\nEnter new value:       ");
               scanf("%f", &newvalue);
               /* Run QC on the new value; allow user to reenter value if */
               /*  it fails QC */
               keeping_newvalue = QCNewValue(newvalue, previous1value,
                     val_code, presentday_month, qc_ptr, opposite_value);}
            *value_to_fix = newvalue;
            return(NEW_VALUE_ENTERED);
            break;
         /* If not a valid choice, ask again */
         default:
            printf("\nYour choice:  ");}}
} /* end FixError */


void GetDay(FILE *raw_fp, struct day *presentday_ptr)
/*-------------------------------------------------------------------------*/
/* Gets line of daily data from raw data file.                             */
/* Receives: *raw_fp:  pointer to raw data file                            */
/*           *presentday_ptr:  address of the daily summary data for the   */
/*                             present day                                 */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: ScanDay                                                          */
/*-------------------------------------------------------------------------*/
{
   /* Read in daily data */
   if(ScanDay(raw_fp, presentday_ptr) == FILE_END){
      printf("\n*** Insufficient data in raw data file ***\n");
      exit(READ_ERROR);}
} /* end GetDay */


int GetDifference(int presentday_jdate, int previousday_jdate,
      int previousday_year)
/*-------------------------------------------------------------------------*/
/* Figures the difference between the present day's date and the previous  */
/*  day's date.                                                            */
/* Receives:  presentday_jdate:  Julian date of the present daily summary  */
/*            previousday_jdate:  Julian date of the previous daily summary*/
/*            previousday_year:  year of the previous daily summary        */
/* Globals: none.                                                          */
/* Returns: the difference                                                 */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int difference; /* difference between the present and previous Julian dates */

   difference = presentday_jdate - previousday_jdate;

   /* If difference is negative, must be due to year change or */
   /*    to an errant present day date */
   if(difference < 0){
      /* If difference is less than -300, likely is a year change */
      if(difference < -300){
         /* Calculate proper difference, taking year change into account */
         if((previousday_year % 4) == 0)
```

```
            difference = DAYSINLYEAR + difference;
         else
            difference = DAYSINYEAR + difference;}}
   return(difference);
}  /* end GetDifference */


void GetHour(FILE *raw_fp, struct hour *presenthour_ptr)
/*-------------------------------------------------------------------------*/
/* Gets line of hourly data from raw data file                             */
/* Receives: *raw_fp:  pointer to raw data file                            */
/*           *presenthour_ptr: address of the data for the present hour    */
/*                             being run through QC                        */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: ScanHour                                                         */
/*-------------------------------------------------------------------------*/
{
   /* Call function to read in hourly data */
   if(ScanHour(raw_fp, presenthour_ptr) == FILE_END){
      printf("\n*** Insufficient data in raw data file ***\n");
      exit(READ_ERROR);}
   return;
}  /* end GetHour */


int GetNewDateFromUser(int previousday_year, int *presentday_year)
/*-------------------------------------------------------------------------*/
/* Lets the user enter a new date.                                         */
/* Receives: previousday_year:  year of the previous daily summary         */
/*           *presentday_year:  address of the year of the present daily   */
/*                              summary                                    */
/* Globals: info                                                           */
/* Returns: julian date of the date entered                                */
/* Calls: JulDate                                                          */
/*        SaveProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
{
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int year; /* two-digit year */
char date_line[11]; /* string to read new date entered by user */
int new_julian_date; /* Julian date of the new date entered by user */
int choice; /* user's choice */

   date_line[0] = 9;

   /* Repeat until a valid date is entered */
   while(TRUE){
      printf("\n\n");
      printf("\tEnter month/day/year (mm/dd/yy):  ");
      /* Get the date */
      cgets(date_line);
      /* See if more than just a return was entered */
      if(date_line[1] != 0){
         /* If something is entered, see if it is in the date format */
         if(sscanf(&date_line[2], "%d/%d/%d", &presentday_month,
               &presentday_day, &year) == 3){
            /* See if a valid date was entered */
            new_julian_date = JulDate(presentday_month, presentday_day, year);
            if((new_julian_date != JD_INVALID_MONTH) && (new_julian_date !=
                  JD_INVALID_DAY) && (new_julian_date != JD_INVALID_YEAR)){
               /* If it is a valid date, see if the year is the same as */
               /*    the previous day's data */
               if(year == previousday_year){
                  /* Return the Julian date */
                  *presentday_year = year;
                  return(new_julian_date);}
               /* See if the new date is in the next calendar year */
               else if(year == previousday_year+1){
                  /* Ask user if the year increment was intended */
                  while(TRUE){
                     printf("\n\tDid you intend to increment the year? ");
```

```
                        choice = getche();
                        if((choice == 'y') || (choice == 'Y')){
                           choice = 'y';
                           break;}
                        if((choice == 'n') || (choice == 'N'))
                           break;}
                     /* See if program info has been updated if year changed */
                     if( (choice == 'y') && (info.year != year) ){
                        /* If not updated, do so */
                        info.year++;
                        SaveProgramInfo();
                        /* Update present day's year also */
                        *presentday_year = year;}

                     /* return the Julian date */
                     return(new_julian_date);}}}}
         /* Prompt user to enter a valid date if it failed any of above tests */
         printf("\n*** Enter a valid date ***\n");}
} /* end GetNewDateFromUser */


int GetNewFirstDateFromUser(int *firstday_year)
/*-------------------------------------------------------------------------*/
/* Gets a new date from the user for the first daily summary collected     */
/* Receives: *firstday_year:  address of the year for the first daily      */
/*                            summary                                      */
/* Globals: none.                                                          */
/* Returns: the julian date of the entered date                            */
/* Calls: JulDate                                                          */
/*-------------------------------------------------------------------------*/
{
int presentday_month; /* month of the present daily summary */
int presentday_day; /* day of the present daily summary */
int year; /* two-digit year */
char date_line[11]; /* string to read new date entered by user */
int new_julian_date; /* Julian date of the new date entered by user */

   date_line[0] = 9;

   /* Repeat until a valid date is entered */
   while(TRUE){
      printf("\n\n");
      printf("\tEnter month/day/year (mm/dd/yy):  ");
      /* Get the date */
      cgets(date_line);
      /* See if just a return was entered */
      if(date_line[1] != 0){
         /* If something is entered, see if it is a date */
         if(sscanf(&date_line[2], "%d/%d/%d", &presentday_month,
               &presentday_day, &year) == 3){
            /* If a valid date was entered, return Julian date */
            new_julian_date = JulDate(presentday_month, presentday_day, year);
            if((new_julian_date != JD_INVALID_MONTH) && (new_julian_date !=
                  JD_INVALID_DAY) && (new_julian_date != JD_INVALID_YEAR)){
               *firstday_year = year;
               return(new_julian_date);}}}
      printf("\n*** Enter a valid date ***\n");}
} /* end GetNewFirstDateFromUser */


void GetPrecip(FILE *raw_fp, int *precip_time, float *precip_amount)
/*-------------------------------------------------------------------------*/
/* Gets line of precip data from raw data file                             */
/* Receives: *raw_fp:  pointer to raw data file                            */
/*           *precip_time:  address of the time of precip occurrance       */
/*           *precip_amount:  address of the amount of precip occurrance   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* Read in precip data */
   if(fscanf(raw_fp,"%d,%f\n", precip_time, precip_amount) == EOF){
```

```
      printf("\n\nError occurred while reading in precip data from raw file");
      exit(READ_ERROR);}
   return;
}  /* end GetPrecip */


void HourQC(struct hour *presenthour_ptr, struct hour *previous1hour_ptr,
      struct hour *previous2hour_ptr, struct qc *qc_ptr)
/*--------------------------------------------------------------------------*/
/* Performs QC procedures on hourly data sets.                              */
/* Receives: *presenthour_ptr:  address of the data of the present hour     */
/*                              being run through QC                        */
/*           *previous1hour_ptr:  address of the hourly summary data for    */
/*                                the hour previous to the present hour     */
/*           *previous2hour_ptr:  address of the hourly summary data for    */
/*                                two hours previous to the present hour    */
/*           *qc_ptr:  address of the qc values                             */
/* Globals: info                                                            */
/* Returns: none.                                                           */
/* Calls: CalDate                                                           */
/*        PrintWarning                                                      */
/*        HourMissingQC                                                     */
/*        HourRHQC                                                          */
/*        HourSRQC                                                          */
/*        HourTempQC                                                        */
/*--------------------------------------------------------------------------*/
{
int warning; /* number signifying the type of QC error found (if any) */
int month_for_showing; /* month of hourly value which failed QC */
int day_for_showing;  /* day of hourly value which failed QC */
int year_for_showing; /* year of hourly value which failed QC */
int HourMissingQC(int, int);
int HourTempQC(float, int, struct qc *, float, float);
int HourSRQC(float, int);
int HourRHQC(float, struct qc *, float, float);

   /* If no previous hours' data (only for very start of collection), signify */
   if(previous1hour_ptr->jdate == NO_DATA)
      previous1hour_ptr->hour = NO_DATA;
   if(previous2hour_ptr->jdate == NO_DATA){
      previous2hour_ptr->temp = (float) NO_DATA;
      previous2hour_ptr->rh = (float) NO_DATA;}


   /* Set the proper year for showing any failed data to user */
   year_for_showing = info.year;
   /* If the current year is a leap year ... */
   if((info.year % 4) == 0){
      /* If the previous day was the last day of the current year, */
      /*    and the previous hour was the last hour of that day... */
      if( (previous1hour_ptr->jdate == 366) &&
            (previous1hour_ptr->hour == 2300) )
         /* Increment year */
         year_for_showing++;}
   /* If the current year is not a leap year ... */
   /* If the previous day was the last day of the current year, */
   /*    and the previous hour was the last hour of that day... */
   else if( (previous1hour_ptr->jdate == 365) &&
         (previous1hour_ptr->hour == 2300) )
      /* Increment year */
      year_for_showing++;

   /* Convert date to month and day */
   if(CalDate(presenthour_ptr->jdate, year_for_showing, &month_for_showing,
         &day_for_showing) == 0){
      /* If date is out of range, print warning */
      PrintWarning(DATEOUTRNG, 0, 0, year_for_showing, presenthour_ptr->hour,
            (float) presenthour_ptr->jdate, VALHDAY, NONE);
      /* Change the date to Jan 1 to avoid problems showing date */
      printf("\nThis day's date is being set to Jan 1, to avoid complications");
      printf("\n\t\t(press any key)");
      getch();
      presenthour_ptr->jdate = 1;
```

```
      /* Skip the rest of the QC procedures */
      return;}

   /* Test the hour to see if it follows the previous hour */
   if((warning = HourMissingQC(presenthour_ptr->hour,
         previous1hour_ptr->hour)) != PASS){
      PrintWarning(warning, month_for_showing, day_for_showing,
            year_for_showing, presenthour_ptr->hour,
            (float) presenthour_ptr->hour, VALHHOU, NONE);
      printf("\t\t(press any key)");
      getch();}

   /* Test the ave temperature */
   if((warning = HourTempQC(presenthour_ptr->temp, month_for_showing, qc_ptr,
         previous1hour_ptr->temp, previous2hour_ptr->temp)) != PASS){
      PrintWarning(warning, month_for_showing, day_for_showing,
            year_for_showing, presenthour_ptr->hour, presenthour_ptr->temp,
            VALHAVT, FRNHT);
      if(warning == AVETOUTRNG)
         printf("   Extreme values:     %6.2f     %6.2f\n",
               qc_ptr->temp[month_for_showing][0],
               qc_ptr->temp[month_for_showing][1]);
      printf("\t\t(press any key)");
      getch();}

   /* Test the ave. solar radiation */
   if((warning = HourSRQC(presenthour_ptr->sr, presenthour_ptr->hour)) != PASS){
      PrintWarning(warning, month_for_showing, day_for_showing,
            year_for_showing, presenthour_ptr->hour, presenthour_ptr->sr,
            VALHTSR, KWPM2);
      printf("\t\t(press any key)");
      getch();}

   /* Test the rel. humidity */
   if((warning = HourRHQC(presenthour_ptr->rh, qc_ptr, previous1hour_ptr->rh,
         previous2hour_ptr->rh)) != PASS){
      PrintWarning(warning, month_for_showing, day_for_showing,
            year_for_showing, presenthour_ptr->hour, presenthour_ptr->rh,
            VALHRLH, PRCNT);
      if(warning == RHOUTRNG)
         printf("   Extreme values:     %6.2f     %6.2f\n",
               qc_ptr->rh[0], qc_ptr->rh[1]);
      printf("\t\t(press any key)");
      getch();}
   return;
}  /* end HourQC */


int HourMissingQC(int presenthour_hour, int previous1hour_hour)
/*-------------------------------------------------------------------------*/
/* Determines if the present hour follows the previous hour by one hour.   */
/* Receives: presenthour_hour:  hour of the present hourly summary         */
/*           previous1hour_hour:  hour of the previous hourly summary      */
/* Globals: none.                                                          */
/* Returns: PASS if hour passes test                                       */
/*          HOURMISSING if hour fails test                                 */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{

   /* If no previous 1 hour's data, just return */
   if(previous1hour_hour == NO_DATA)
      return(PASS);

   /* Take midnight into account */
   if(presenthour_hour == 0)
      if(previous1hour_hour != 2300)
         return(HOURMISSING);
      else
         return(PASS);

   /* For all other hours of the day */
   if(presenthour_hour != previous1hour_hour + ONEHOUR)
```

```
      return(HOURMISSING);
   else
      return(PASS);
}  /* end HourMissingQC */


int HourRHQC(float presenthour_rh, struct qc *qc_ptr, float previous1hour_rh,
      float previous2hour_rh)
/*-------------------------------------------------------------------------*/
/* Performs QC procedures on hourly rel. humidity values.                  */
/* Receives:  presenthour_rh: rel. hum. value of the present hourly summary*/
/*            *qc_ptr:  address of QC values                               */
/*            previous1hour_rh:  rel. hum. value of previous hourly summary*/
/*            previous2hour_rh:  rel. hum. value of 2-hours previous       */
/* Returns: PASS if value passes test                                      */
/*          RHOUTRNG if value is out of range                              */
/*          RHCONSTNT if value is constant over time                       */
/* Calls: CheckConstant                                                    */
/*        CheckExtreme                                                     */
/*-------------------------------------------------------------------------*/
{
   /* Make sure RH is within range */
   if(CheckExtreme(presenthour_rh, qc_ptr->rh) != PASS)
         return(RHOUTRNG);
   /* Make sure RH is not exactly the same as previous hours' */
   if(previous2hour_rh < (float) NO_DATA)
      if(CheckConstant(presenthour_rh, previous1hour_rh, previous2hour_rh)
            != PASS)
         return(RHCONSTNT);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end HourRHQC */


int HourSRQC(float presenthour_sr, int presenthour_hour)
/*-------------------------------------------------------------------------*/
/* Performs QC procedures on hourly ave. solar radiation value.            */
/* Receives: presenthour_sr:  solar rad. value of present hourly summary   */
/*           presenthour_hour:  hour of present hourly summary             */
/* Globals: none.                                                          */
/* Returns: PASS if value passes all tests                                 */
/*          SOLARPOSITIV if value is positive at night                     */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* Make sure SR is not positive at night (midnight to 4 am) */
   if(presenthour_hour <= NIGHT)
      if(presenthour_sr > 0.0)
         return(SOLARPOSITIV);
   /* If not night or not positive at night, return passing value */
   return(PASS);
}  /* end HourSRQC */


int HourTempQC(float presenthour_temp, int presenthour_month,
      struct qc *qc_ptr, float previous1hour_temp, float previous2hour_temp)
/*-------------------------------------------------------------------------*/
/* Performs QC procedures on hourly ave. temperature value.                */
/* Receives:  presenthour_temp:  ave. temp. value of present hourly summary*/
/*            presenthour_month:  month of present hourly summary          */
/*            *qc_ptr:  address of QC values                               */
/*            previous1hour_temp:  ave. temp. value of previous hourly     */
/*                                 summary                                 */
/*            previous2hour_temp:  ave. temp. value of 2-hours previous    */
/* Returns: PASS if value passes test                                      */
/*          AVETOUTRNG if value is out of range                            */
/*          AVETCONSTNT if value is constant over time                     */
/* Calls: CheckConstant                                                    */
/*        CheckExtreme                                                     */
/*-------------------------------------------------------------------------*/
{
   /* Make sure temp is within range (different values for each month) */
   if(CheckExtreme(presenthour_temp, qc_ptr->temp[presenthour_month]) != PASS)
```

```
      return(AVETOUTRNG);
   /* Make sure temp is not exactly the same as previous hours' */
   if(previous2hour_temp < (float) NO_DATA)
      if(CheckConstant(presenthour_temp, previous1hour_temp,
            previous2hour_temp) != PASS)
         return(AVETCONSTNT);
   /* If no check failures, return passing value */
   return(PASS);
}  /* end HourTempQC */


FILE *OpenCurrentFile(void)
/*-------------------------------------------------------------------------*/
/* Opens file of current weather data                                      */
/* Receives: none.                                                         */
/* Globals: info                                                           */
/* Returns: file pointer to the opened file                                */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
FILE *current_fp;  /* pointer to the current data file */

   /* Open to append file */
   /* If file doesn't exist, create and put in first line of IBSNAT codes */
   if((current_fp = fopen(info.current_file, READ)) == NULL){
      current_fp = fopen(info.current_file, APPEND);
      fprintf(current_fp, "%2s%2s %6.2f %6.2f %5.2f %5.2f\n", info.location_id,
            info.station_id, info.lat, info.lon, info.parfac,
            info.pardat);}
   else
      current_fp = fopen(info.current_file, APPEND);
   return(current_fp);
}  /* end OpenCurrentFile */


FILE *OpenPreviousFile(struct day *previousday_ptr,
      struct hour *previous1hour_ptr, struct hour *previous2hour_ptr)
/*-------------------------------------------------------------------------*/
/* Opens file with previous data.                                          */
/* Receives: *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           *previous1hour_ptr:  address of the hourly summary data for   */
/*                                the hour previous to the present hour    */
/*           *previous2hour_ptr:  address of the hourly summary data for   */
/*                                two hours previous to the present hour   */
/* Globals: info                                                           */
/* Returns: file pointer to the opened file                                */
/* Calls: ScanDay                                                          */
/*        ScanHour                                                         */
/*-------------------------------------------------------------------------*/
{
FILE *previous_fp; /* pointer to the previous data file */

   /* The following if statements provide for time when data collection is
       just beginning */
   /* If file does not exist, create and signify no data in arrays */
   if((previous_fp = fopen(info.prev_file, READWRITE)) == NULL){
      previous_fp = fopen(info.prev_file, WRITE);
      previous1hour_ptr->jdate = NO_DATA;
      previous2hour_ptr->jdate = NO_DATA;
      previousday_ptr->jdate = NO_DATA;
      return(previous_fp);}
   /* If no hourly record in file, say so; else read in data */
   if(ScanHour(previous_fp, previous1hour_ptr) == FILE_END){
      previous1hour_ptr->jdate = NO_DATA;
      previous2hour_ptr->jdate = NO_DATA;
      previousday_ptr->jdate = NO_DATA;
      return(previous_fp);}
   /* If no record for 2 hours prior, say so; else read in data */
   if(ScanHour(previous_fp, previous2hour_ptr) == FILE_END){
      previous2hour_ptr->jdate = NO_DATA;
      previousday_ptr->jdate = NO_DATA;
```

```
      return(previous_fp);}
   /* If no previous day record, say so; else read in data */
   if(ScanDay(previous_fp, previousday_ptr) == FILE_END){
      previousday_ptr->jdate = NO_DATA;
      return(previous_fp);}
   return(previous_fp);
}  /* end OpenPreviousFile */


void PrintArrayID(FILE *backup_fp, int arrayid)
/*-------------------------------------------------------------------------*/
/* Prints the array ID to the back-up file.                                */
/* Receives: *backup_fp: pointer to the back-up file                       */
/*           arrayid:  ID number for hourly, daily, or precip occurrance   */
/*                     summary                                             */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   fprintf(backup_fp, "%d,", arrayid);
   return;
}  /* end PrintArrayID */


void PrintJulianDateWarning(int warning, struct day *presentday_ptr,
      int presentday_month, int presentday_day, int previousday_month,
      int previousday_day, int previousday_year)
/*-------------------------------------------------------------------------*/
/* Prints a warning for a Julian date QC error to the screen.              */
/* Receives: warning:  number signifying the type of QC error (if any)     */
/*           *presentday_ptr:  address of the data of the present day      */
/*                             being run through QC                        */
/*           presentday_month:  month of the present daily summary         */
/*           presentday_day:  day of the present daily summary             */
/*           previousday_month:  month of the previous daily summary       */
/*           previousday_day:  day of the present daily summary            */
/*           previousday_year:  year of the previous daily summary         */
/* Globals: error_code                                                     */
/*          month_name                                                     */
/* Returns: none.                                                          */
/* Calls: PrintDaysDataToScreen                                            */
/*-------------------------------------------------------------------------*/
{
   /* Print message */
   printf("\n\n\n");
   printf("\t--- Report from:  QC procedures on raw data ");
   printf("from weather station ---");
   printf("\n\nWARNING:  %s\n", error_code[warning]);
   printf("   Recorded date:  ");
   if(warning == DATEOUTRNG)
      printf("[invalid date]\n");
   else
      printf("%s %d, %d\n", month_name[presentday_month], presentday_day,
            presentday_ptr->year);
   printf("\tSummary of this day's data:\n");
   /* Print summary of present day's data */
   PrintDaysDataToScreen(presentday_ptr);
   printf("\nLast daily summary in current data file is for %s %d, %d\n",
         month_name[previousday_month], previousday_day, previousday_year);
}  /* end PrintJulianDateWarning */


void PrintPrecip(FILE *fp, int precip_time, float precip_amount)
/*-------------------------------------------------------------------------*/
/* Prints precip data to the back-up file.                                 */
/* Receives: fp: file pointer to the back-up file                          */
/*           precip_time: time of the precip occurrance                    */
/*           precip_amount: amount of the precip occurrance                */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
```

```
{
   fprintf(fp,"%d,%.3f\n", precip_time, precip_amount);
   return;
} /* end PrintPrecip */


void PrintWarning(int warning, int month, int day, int year, int hour,
      float warning_value, int val_code, int unit_code)
/*-------------------------------------------------------------------------*/
/* Prints a warning for a QC error to the screen.                          */
/* Receives: warning:  code signifying the QC error                        */
/*           month:  month number (1-12)                                   */
/*           day:  day of month                                            */
/*           year:  two-digit year                                         */
/*           hour:  hour in 24-hour format                                 */
/*           warning_value:  value which caused the QC error               */
/*           val_code:  number signifying what the value represents        */
/*           unit_code:  code signifying units for a value                 */
/* Globals: value_code                                                     */
/*          error_code                                                     */
/*          units_code                                                     */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* Print message */
   printf("\n\n\n");
   printf("\t--- Report from:  QC procedures on raw data ");
   printf("from weather station ---");
   printf("\n\nWARNING:  %s\n", error_code[warning]);
   printf("   Recorded date:  ");
   if(month == 0)
      printf("[invalid day], %d hours\n", hour);
   else
      printf("%s %d, %d, %d hours\n", month_name[month], day, year, hour);
   /* Decide whether to print a float value or int value */
   if((val_code == VALHHOU) || (val_code == VALHDAY))
      printf("   Errant value:       %d  ", (int) warning_value);
   else
      printf("   Errant value:       %6.2f ", warning_value);
   /* Print the units and name of the value */
   printf("%s  %s\n", units_code[unit_code], value_code[val_code]);
} /* end PrintWarning */


void SavePrevious(FILE *previous_fp, struct day *previousday_ptr,
      struct hour *previous1hour_ptr, struct hour *previous2hour_ptr)
/*-------------------------------------------------------------------------*/
/* Saves data used for future qc in previous file.                         */
/* Receives: previous_fp: pointer to the previous data file                */
/*           *previousday_ptr:  address of the collected daily summary     */
/*                              data for the day previous to the present   */
/*                              day being run through QC                   */
/*           *previous1hour_ptr:  address of the hourly summary data for   */
/*                                the hour previous to the present hour    */
/*           *previous2hour_ptr:  address of the hourly summary data for   */
/*                                two hours previous to the present hour   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: PrintDay                                                         */
/*        PrintHour                                                        */
/*-------------------------------------------------------------------------*/
{
   /* Move to beginning of previous file */
   rewind(previous_fp);
   /* Print 1 hour prior data */
   PrintHour(previous_fp, previous1hour_ptr);
   /* Print 2 hours prior data */
   PrintHour(previous_fp, previous2hour_ptr);
   /* Print 1 day prior data */
   PrintDay(previous_fp, previousday_ptr);
   return;
} /* end SavePrevious */
```

```
int ScanArrayID(FILE *raw_fp)
/*-------------------------------------------------------------------------*/
/* Reads the array ID from the raw data file.                              */
/* Receives: *raw_fp: pointer to the raw data file                         */
/* Globals: none.                                                          */
/* Returns: the ID number read                                             */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int arrayid;

   if(fscanf(raw_fp,"%d,", &arrayid) == EOF)
      return(FILE_END);
   else
      return(arrayid);
}  /* end ScanArrayID */
```

WEDIT.C

```
/* Include files */
#include <conio.h>      /* for console functions */
#include <stdio.h>      /* for file functions */
#include <graph.h>      /* for clearscreen function */
#include <process.h>    /* for exit function */
#include "whdr.h"       /* defines, structs, functions for this program */
#include "wglobal.h"    /* global variables for this program */



void FileViewEdit(void)
/*-------------------------------------------------------------------------*/
/* Allows user to edit current and QC files and view current, current/     */
/*    gen'd, back-up, and QC files.                                        */
/* Receives: none.                                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: FileViewEditMenu                                                 */
/*        ShowFile                                                         */
/*        GetViewEditAction                                                */
/*        EditCurrentDataFile                                              */
/*        EditQCValues                                                     */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */
int view_or_edit;  /* indicates user's choice to view or edit a file */

/* Functions */
void FileViewEditMenu(void);
char GetViewEditAction(void);
void ShowFile(char *);
void EditCurrentDataFile(void);
void EditQCValues(void);

   while(TRUE){
      /* Show menu and get user's choice */
      _clearscreen(_GCLEARSCREEN);
      FileViewEditMenu();
      choice = getche();
      if(choice == '5')
         break;
      switch(choice){
         case '1':
            view_or_edit = GetViewEditAction();
            if(view_or_edit == 'v')
               ShowFile(info.current_file);
            if(view_or_edit == 'e')
               EditCurrentDataFile();
            break;
         case '2':
            ShowFile(info.curgen_file);
            break;
         case '3':
            view_or_edit = GetViewEditAction();
            if(view_or_edit == 'v')
               ShowFile(info.qc_file);
            if(view_or_edit == 'e')
               EditQCValues();
            break;
         case '4':
            ShowFile(info.backup_file);
            break;
         default:
            printf("\n\n*** Enter 1-5 ***\n");
            getch();}}

} /* end FileViewEdit */
```

```
void FileViewEditMenu(void)
/*-------------------------------------------------------------------------*/
/* Shows user choice of files to view and/or edit.                         */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   printf("\n\n\n\t\t\tMENU FOR VIEW/EDIT OPTIONS\n\n\n");
   printf("\t\t1.  View/Edit File of Current Data\n\n");
   printf("\t\t2.  View File of Current and Generated Data\n\n");
   printf("\t\t3.  View/Edit File of QC Values\n\n");
   printf("\t\t4.  View File of Weather Station Back-up Data\n\n");
   printf("\t\t5.  Return to Main Menu\n\n\n\n");
   printf("\t\tYour Choice:  ");
}  /* end FileViewEditMenu */



/* Other functions */


float ConvertForQC(float value_to_convert, int val_code)
/*-------------------------------------------------------------------------*/
/* Converts SI units to English units for temperature and precip.          */
/* Receives:  value_to_convert:  value to be converted to proper units for */
/*                               QC tests                                  */
/*            val_code:  number signifying what the value represents       */
/* Globals:  none.                                                         */
/* Returns:  the converted float value                                     */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
float converted_value; /* value converted to proper units for QC tests */

   /* If a temperature value, convert from C to F */
   if((val_code == VALDMXT) || (val_code == VALDMNT))
      converted_value = (value_to_convert / FTOC59) + FTOC32;
   /* If a precip. value, convert from cm to in */
   else if(val_code == VALDTPR)
      converted_value = value_to_convert / INTOCM;
   /* If a solar rad. value, units of MJ/m2 don't need converting */
   else if(val_code == VALDTSR)
      converted_value = value_to_convert;
   /* Return converted value */
   return(converted_value);
}  /* end ConvertForQC */


void EditCurrentDataFile(void)
/*-------------------------------------------------------------------------*/
/* Lets user change data stored in the current data file.                  */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: EditDayOfCurrentData                                             */
/*        JulDate                                                          */
/*-------------------------------------------------------------------------*/
{
int month; /* month number (1-12) */
int day; /* day of month */
int year; /* two-digit year */
char date_line[11]; /* string to read new date entered by user */
int day_to_edit; /* Julian date of daily summary that user wants to edit */
void EditDayOfCurrentData(int, int, int, int);

   date_line[0] = 9;

   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\n\t(Press 'enter' to exit)\n\n");
```

```
      printf("\tEnter month/day/year of data to change (mm/dd/yy):  ");
      cgets(date_line);
      /* See if just a return was entered */
      if(date_line[1] != 0){
         /* If something is entered, see if it is a date */
         if(sscanf(&date_line[2], "%d/%d/%d", &month, &day,
               &year) == 3){
            /* If a valid date was entered, let user change the day's data */
            day_to_edit = JulDate(month, day, year);
            if((day_to_edit != JD_INVALID_MONTH) && (day_to_edit !=
                  JD_INVALID_DAY) && (day_to_edit != JD_INVALID_YEAR))
               EditDayOfCurrentData(day_to_edit, month, day, year);}}
      else
         /* Exit if a return was entered */
         return;}
} /* end EditCurrentDataFile */


void EditDayOfCurrentData(int day_to_edit, int month, int day, int year)
/*-------------------------------------------------------------------------*/
/* Lets user change a day's data.                                          */
/* Receives: day_to_edit:  Julian date of daily summary user wants to edit */
/*           month:  month number (1-12)                                   */
/*           day:  day of month                                            */
/*           year:  two-digit year                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: GetValueToChange                                                 */
/*        OpenFile                                                         */
/*        GetFormattedDay                                                  */
/*        PrintFormattedDay                                                */
/*        GetQCValues                                                      */
/*        GetNewValue                                                      */
/*-------------------------------------------------------------------------*/
{
int open_result; /* signifies if error occurred during file opening */
int day_to_edit_found = FALSE; /* indicates if daily summary that user wants */
                               /*    to edit has been located in the current */
                               /*    data file */
FILE *current_fp; /* pointer to current data file */
FILE *temporary_fp; /* pointer to temporary data file */
char location_id[3]; /* string for location ID (for IBSNAT format) */
char station_id[3]; /* string for station ID (for IBSNAT format) */
float latitude; /* latitude of weather station (for IBSNAT format) */
float longitude; /* longitude of weather station (for IBSNAT format) */
float parfac; /* factor for converting MJ/M2 to PAR (for IBSNAT format) */
float pardat; /* flag indicating if PAR is included in weather file (for */
              /*    IBSNAT format */
struct formatted_day current_daydata; /* IBSNAT-format daily summary from */
                                      /*    current file */
struct qc qcvalues; /* values for QC tests */
int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION]; /* array of QC */
                                                           /* errors and actions */
int value_to_change;/* user's choice of which daily summary value to change */
int GetValueToChange(struct formatted_day *, int, int, int);
float GetNewValue(int, int, int, struct qc *);

   /* Open daily and temp file */
   current_fp = OpenFile(&open_result, info.current_file, READ);
   if(open_result == ERROR){
      printf("\nCurrent data file does not exist!\n");
      printf("\t(press any key)");
      getch();
      return;}
   temporary_fp = OpenFile(&open_result, TEMP_FILE, WRITE);
   /* Skip past line of header data in file */
   if((fscanf(current_fp, "%2s%2s %f %f %f %f\n", location_id, station_id,
         &latitude, &longitude, &parfac, &pardat)) == EOF){
      /* Stop if no data in file */
      printf("\n\tNo data in formatted file!\n\t(press any key)");
      getch();
      fcloseall();
      remove(TEMP_FILE);
```

```
   return;}
/* Write header to temp file */
fprintf(temporary_fp, "%2s%2s %6.2f %6.2f %5.2f %5.2f\n", location_id,
      station_id, latitude, longitude, parfac, pardat);
/* Read in a line of data until EOF */
while(GetFormattedDay(current_fp, &current_daydata, erroractions) !=
      FILE_END){
   /* See if this day is the day of data to edit */
   if(current_daydata.fjdate != day_to_edit)
      /* If not the day of data to edit, put in temp file */
      PrintFormattedDay(temporary_fp, &current_daydata, erroractions);
   /* Make sure the year of the data matches the year the user entered */
   else if(current_daydata.fyear == year){
      /* Have found the day of data to edit */
      day_to_edit_found = TRUE;
      /* Load QC values */
      if(GetQCValues(&qcvalues) == ERROR){
         printf("\nError occured while reading qc data file\n");
         exit(QC_FILE_READ_ERROR);}
      /* Let user edit each data value until 'quit' is chosen */
      while(TRUE){
         value_to_change = GetValueToChange(&current_daydata, month, day,
               year);
         /* Get out of loop if user chose to quit */
         if(value_to_change == 0)
            break;
         switch(value_to_change){
            case 1:
               /* Get the new value from the user */
               current_daydata.fmaxt = GetNewValue(VALDMXT, CLCIS, month,
                     &qcvalues);
               /* Update error/action codes to reflect the edit */
               erroractions[MAXTEMP][QC_ACTION] = NEW_VALUE_ENTERED;
               if(erroractions[MAXTEMP][QC_ERROR] == 0)
                  erroractions[MAXTEMP][QC_ERROR] = MAXTEDITED;
               break;
            case 2:
               /* Get the new value from the user */
               current_daydata.fmint = GetNewValue(VALDMNT, CLCIS, month,
                     &qcvalues);
               /* Update error/action codes to reflect the edit */
               erroractions[MINTEMP][QC_ACTION] = NEW_VALUE_ENTERED;
               if(erroractions[MINTEMP][QC_ERROR] == 0)
                  erroractions[MINTEMP][QC_ERROR] = MINTEDITED;
               break;
            case 3:
               /* Get the new value from the user */
               current_daydata.fsr = GetNewValue(VALDTSR, MJPDA, month,
                     &qcvalues);
               /* Update error/action codes to reflect the edit */
               erroractions[SOLAR][QC_ACTION] = NEW_VALUE_ENTERED;
               if(erroractions[SOLAR][QC_ERROR] == 0)
                  erroractions[SOLAR][QC_ERROR] = SOLAREDITED;
               break;
            case 4:
               /* Get the new value from the user */
               current_daydata.fprec = GetNewValue(VALDTPR, CMETR, month,
                     &qcvalues);
               /* Update error/action codes to reflect the edit */
               erroractions[PRECIP][QC_ACTION] = NEW_VALUE_ENTERED;
               if(erroractions[PRECIP][QC_ERROR] == 0)
                  erroractions[PRECIP][QC_ERROR] = PRECIPEDITED;
               break;}}
      PrintFormattedDay(temporary_fp, &current_daydata, erroractions);}
   /* If the year was wrong, save this day's data and save rest of file */
   else
      /* If not the day of data to edit, put in temp file */
      PrintFormattedDay(temporary_fp, &current_daydata, erroractions);}

if(day_to_edit_found == FALSE){
   printf("\n*** Could not find that day ***\n");
   getch();}
fcloseall();
```

```
   remove(info.current_file);
   rename(TEMP_FILE, info.current_file);
   return;
}  /* end EditDayOfCurrentData */


void EditQCValues(void)
/*-------------------------------------------------------------------------*/
/* Allows user to edit QC values file.                                     */
/* Receives: none.                                                         */
/* Globals: units_code                                                     */
/* Returns: none.                                                          */
/* Calls: GetQCValues                                                      */
/*        ZeroQCValues                                                     */
/*        EnterQCMaxmin                                                    */
/*        SaveQCValues                                                     */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */
int finished = FALSE; /* indicates when user is finished editing */
struct qc qcvalues; /* values for QC tests */
int i; /* loop counter */
char *qc_measurement[6] = {"temperature",
                           "solar radiation",
                           "precipitation",
                           "relative humidity",
                           "datalogger voltage",
                           "datalogger moisture",};
            /* string array of the names of measurements that have QC values */
/* Values to access labels in qc_measurement[] */
const int temp = 0;
const int sr = 1;
const int prec = 2;
const int rh = 3;
const int volt = 4;
const int mois = 5;
char entered_string[9]; /* used to get value entered by user */
float entered_value; /* numeric value from the string entered by user */

/* Local functions */
void ZeroQCValues(struct qc *);
void EnterQCMaxmin(float [MAXMIN], char *, char *);
void SaveQCValues(struct qc *);

   entered_string[0] = 7;

   /* Ask if want to proceed */
   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\t\tEditor for Quality Control Values\n\n");
      printf("Do you want to proceed with the edit? ");
      choice = getche();
      if((choice == 'y') || (choice == 'Y')){
         printf("\n\n");
         break;}
      if((choice == 'n') || (choice == 'N'))
         return;}
   /* Load the current_daydata values; if none, set all to zero */
   if(GetQCValues(&qcvalues) != OK)
      ZeroQCValues(&qcvalues);

   /* Display values and descriptions and allow user to edit until */
   /*    done editing */
   while(finished == FALSE){
      printf("\tExisting or default values are given in brackets.\n");
      printf("\tPress 'enter' to keep existing value.\n\n");
      for(i=1; i<=MONTHS; i++){
         printf("\n%s\n", month_name[i]);
         EnterQCMaxmin(qcvalues.temp[i], qc_measurement[temp],
               units_code[FRNHT]);}
      for(i=1; i<=MONTHS; i++){
         printf("\n%s\n", month_name[i]);
         EnterQCMaxmin(qcvalues.sr[i], qc_measurement[sr],
```

```
                units_code[MJPDA]);}
        for(i=1; i<=MONTHS; i++){
           printf("\n%s\n", month_name[i]);
           EnterQCMaxmin(qcvalues.prec[i], qc_measurement[prec],
                units_code[INCHS]);}

        printf("\nThe following values only require one maximum and minimum:\n");
        EnterQCMaxmin(qcvalues.rh, qc_measurement[rh], units_code[PRCNT]);
        printf("\n");
        EnterQCMaxmin(qcvalues.volt, qc_measurement[volt], units_code[VOLTS]);
        printf("\n");
        EnterQCMaxmin(qcvalues.mois, qc_measurement[mois], units_code[INDEX]);

        printf("\nEnter the maximum allowable daily change \n");
        printf("\tin temperature %s:  [%6.2f]  ", units_code[FRNHT],
              qcvalues.maxtempchng);
        /* Repeat until get a valid answer */
        while(TRUE){
           /* Get input from user */
           cgets(entered_string);
           /* See if just a return was entered */
           if(entered_string[1] != 0){
              /* If a number entered, set value equal to the number */
              if(sscanf(&entered_string[2], "%f", &entered_value) != 0){
                 qcvalues.maxtempchng = entered_value;
                 break;}}
           /* If a return was entered, don't change value */
           else
              break;}

        while(TRUE){
           printf("\nDo you need to reenter any of these values?");
           choice = getche();
           if((choice == 'n') || (choice == 'N')){
              finished = TRUE;
              break;}
           if((choice == 'y') || (choice == 'Y'))
              break;}}

     SaveQCValues(&qcvalues);
     return;
}  /* end EditQCValues */


void EnterQCMaxmin(float qc_data[MAXMIN], char *measurement,
        char *measurement_units)
/*-------------------------------------------------------------------------*/
/* Lets user enter one max and one min value.                              */
/* Receives: qc_maxmin[]:  array with max and min values in it             */
/*           *measurement:  address of string containing name of current   */
/*                          measurement                                    */
/*           *measurement_units:  address of string containing units of    */
/*                                current QC measurement                   */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
char *maxmin[MAXMIN] = {"maximum",    /* string array with words to identify */
                        "minimum");   /*    max and min values */
int i; /* loop counter */
char entered_string[9]; /* used to get value entered by user */
float entered_value; /* numeric value from the string entered by user */

   entered_string[0] = 7;

   for(i=0; i<=MAXMIN-1; i++){
      printf("\t%s ", maxmin[i]);
      printf("%s %s:  [%6.2f]  ", measurement, measurement_units, qc_data[i]);
      /* Repeat until get a valid answer */
      while(TRUE){
         /* Get input from user */
         cgets(entered_string);
```

```
         /* See if just a return was entered */
         if(entered_string[1] != 0){
            /* If a number entered, set value equal to the number */
            if(sscanf(&entered_string[2], "%f", &entered_value) != 0){
               qc_data[i] = entered_value;
               break;}}
         /* If a return was entered, don't change value */
         else
            break;}
      printf("\n");}
   return;
} /* end EnterQCMaxmin */


int GetDataLineFromFile(FILE *fp, char line_from_file[100])
/*-------------------------------------------------------------------------*/
/* Gets line of data from a file as a string.                              */
/* Receives: *fp:  pointer to file                                         */
/*           line_from_file[]:  string to read a line of data from file    */
/* Globals: none.                                                          */
/* Returns: FILE_END:  if end of file is reached                           */
/*          OK:  if line of data was read                                  */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
const maxfgets = 99;     /* Maximum length of a data line that is read */

   /* Get line of data or quit if at end of file */
   if((fgets(line_from_file, maxfgets, fp)) != NULL)
      return(OK);
   else
      return(FILE_END);
} /* end GetDataLineFromFile */


float GetNewValue(int val_code, int unit_code, int month, struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Prints the name and units of the value to enter, gets a float value     */
/*  from user, converts it to proper units for the qc tests, runs qc       */
/*  procedures on it, repeats this if value fails qc and user wants to     */
/*  reenter it                                                             */
/* Receives:  val_code:  number signifying what the value represents       */
/*            unit_code:  code signifying units for a value                */
/*            month:  month number (1-12)                                  */
/*            *qcptr:  address of the qc values                            */
/* Globals:  value_code                                                    */
/*           units_code                                                    */
/* Returns:  the float value entered                                       */
/* Calls:  ConvertForQC                                                    */
/*         QCNewValue                                                      */
/*-------------------------------------------------------------------------*/
{
float newvalue;  /* new value entered by user for one which failed QC */
float converted_newvalue; /* new value converted to proper units for qc tests */
int keeping_newvalue = FAIL;  /* indicates if user wants to keep or re-enter */
                              /*    the new value that failed QC tests */
float ConvertForQC(float, int);

   /* Get a value until the value is valid */
   while(keeping_newvalue == FAIL){
      /* Print value name and units and get the value from user */
      printf("\t%s %s:  ", value_code[val_code], units_code[unit_code]);
      scanf("%f", &newvalue);
      /* Convert value to units used in qc tests */
      converted_newvalue = ConvertForQC(newvalue, val_code);
      /* Run qc tests and return PASS or FAIL */
      keeping_newvalue = QCNewValue(converted_newvalue, (float) NO_DATA,
            val_code, month, qcptr, (float) NO_DATA);}
   /* return unconverted new value */
   return(newvalue);
} /* end GetNewValue */
```

```
int GetValueToChange(struct formatted_day *formatted_ptr, int month, int day,
     int year)
/*-------------------------------------------------------------------------*/
/* Lets user choose which value of the daily data to edit.                 */
/* Receives: *formatted_ptr:  address of the formatted data                */
/*           month:  month number (1-12)                                   */
/*           day:  day of month                                            */
/*           year:  two-digit year                                         */
/* Globals: none.                                                          */
/* Returns: integer (1-4):  corresponding to value to view                 */
/*          integer value of q or Q:  indicating user wants to quit editing*/
/* Calls: ShowValueChoices                                                 */
/*-------------------------------------------------------------------------*/
{
int choice;   /* user's choice */
int ascii_number;      /* ASCII value of the user's choice */
void ShowValueChoices(struct formatted_day *, int, int, int);

   while(TRUE){
      /* Show choices of values to choose from and get users choice */
      _clearscreen(_GCLEARSCREEN);
      ShowValueChoices(formatted_ptr, month, day, year);
      /* Get choice */
      choice = getche();
      printf("\n\n\n");
      /* Quit if user wants */
      if((choice == 'q') || (choice == 'Q')){
         ascii_number = 0;
         return(ascii_number);}
      /* If a valid number, convert its ASCII value to the proper choice */
         switch(choice){
            case '1':
            case '2':
            case '3':
            case '4':
               ascii_number = choice - 48;
               /* Return number of value to view */
               return(ascii_number);
               break;
            default:
               printf("\n\t\t*** Enter correct value ***\n");
               getch();}}
}  /* end GetValueToChange */


char GetViewEditAction(void)
/*-------------------------------------------------------------------------*/
/* Gets user's decision to edit or view a file, or exit to the previous    */
/*  menu.                                                                  */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: v:  if user wants to view                                      */
/*          e:  if user wants to edit                                      */
/*          x:  if user wants to exit (return to previous menu)            */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int view_or_edit; /* indicates user's choice to view or edit a file */

   /* Ask if want to view or edit the file */
   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\n\n\n\n\tDo you want to view or edit this file?");
      printf("\n\t\t(v) view; (e) edit; (x) exit:  ");
      view_or_edit = getche();
      if((view_or_edit == 'v') || (view_or_edit == 'V'))
         return('v');
      if((view_or_edit == 'e') || (view_or_edit == 'E'))
         return('e');
      if((view_or_edit == 'x') || (view_or_edit == 'X'))
         return('x');}

}  /* end GetViewEditAction */
```

```
void SaveQCValues(struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Saves QC values in a file.                                              */
/* Receives: *qcptr:  address of qc values                                 */
/* Globals: info                                                           */
/* Returns:  none.                                                         */
/* Calls:  none.                                                           */
/*-------------------------------------------------------------------------*/
{
int i; /* loop counter */
int j = 0; /* loop counter */
FILE *qc_fp; /* pointer to QC values file */
int open_result; /* signifies if error occurred during file opening */

   /* Open QC values file */
   qc_fp = OpenFile(&open_result, info.qc_file, WRITE);
   if(open_result == ERROR){
      printf("\nError occured while saving qc file\n");
      exit(QC_FILE_WRITE_ERROR);}

   /* For each month, put temp values in file */
   for(i=1; i<=MONTHS; i++){
      /* Put ID for this set of values on same line as first two values */
      if(i == 1){
         if(fprintf(qc_fp,"%6.2f,%6.2f     (temperature)\n",
               qcptr->temp[i][j], qcptr->temp[i][j+1]) == NULL){
            printf("\nError occured while saving qc file\n");
            exit(QC_FILE_WRITE_ERROR);}}
      /* After first two values, just put two values on each line */
      else if(fprintf(qc_fp,"%6.2f,%6.2f\n", qcptr->temp[i][j],
            qcptr->temp[i][j+1]) == NULL){
         printf("\nError occured while saving qc file\n");
         exit(QC_FILE_WRITE_ERROR);}}

   /* For each month, put solar values in file */
   for(i=1; i<=MONTHS; i++){
      /* Put ID for this set of values on same line as first two values */
      if(i == 1){
         if(fprintf(qc_fp,"%6.2f,%6.2f     (solar-radiation)\n",
               qcptr->sr[i][j], qcptr->sr[i][j+1]) == NULL){
            printf("\nError occured while saving qc file\n");
            exit(QC_FILE_WRITE_ERROR);}}
      /* After first two values, just put two values on each line */
      else if(fprintf(qc_fp,"%6.2f,%6.2f\n", qcptr->sr[i][j],
            qcptr->sr[i][j+1]) == NULL){
         printf("\nError occured while saving qc file\n");
         exit(QC_FILE_WRITE_ERROR);}}

   /* For each month, put precip values in file */
   for(i=1; i<=MONTHS; i++){
      /* Put ID for this set of values on same line as first two values */
      if(i == 1){
         if(fprintf(qc_fp,"%6.2f,%6.2f     (precipitation)\n",
               qcptr->prec[i][j], qcptr->prec[i][j+1]) == NULL){
            printf("\nError occured while saving qc file\n");
            exit(QC_FILE_WRITE_ERROR);}}
      /* After first two values, just put two values on each line */
      else if(fprintf(qc_fp,"%6.2f,%6.2f\n", qcptr->prec[i][j],
            qcptr->prec[i][j+1]) == NULL){
         printf("\nError occured while saving qc file\n");
         exit(QC_FILE_WRITE_ERROR);}}

   /* Put RH values in file with ID on same line */
   if(fprintf(qc_fp,"%6.2f,%6.2f     (relative-humidity)\n", qcptr->rh[j],
         qcptr->rh[j+1]) == NULL){
      printf("\nError occured while saving qc file\n");
      exit(QC_FILE_WRITE_ERROR);}

   /* Put voltage values in file with ID on same line */
   if(fprintf(qc_fp,"%6.2f,%6.2f     (datalogger-voltage)\n", qcptr->volt[j],
         qcptr->volt[j+1]) == NULL){
      printf("\nError occured while saving qc file\n");
```

```
      exit(QC_FILE_WRITE_ERROR);}

   /* Put moisture values in file with ID on same line */
   if(fprintf(qc_fp,"%6.2f,%6.2f     (datalogger-moisture)\n",qcptr->mois[j],
         qcptr->mois[j+1]) == NULL){
      printf("\nError occured while saving qc file\n");
      exit(QC_FILE_WRITE_ERROR);}

   /* Put temp change value in file with ID on same line */
   if(fprintf(qc_fp, "%6.2f          (temperature-change)\n",
         qcptr->maxtempchng) == NULL){
      printf("\nError occured while saving qc file\n");
      exit(QC_FILE_WRITE_ERROR);}

   fclose(qc_fp);
   return;
} /* end SaveQCValues */


void ShowFile(char *file_name_to_view)
/*-------------------------------------------------------------------------*/
/* Puts file contents on screen or printer, line by line.                  */
/* Receives: *file_name_to_view:  address of name of file to view          */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: OpenFile                                                         */
/*        GetDataLineFromFile                                              */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */
FILE *view_in_fp;  /* Pointer to data file to view */
FILE *view_out_fp; /* Pointer to output file for viewing (screen or printer) */
int screen_view = TRUE; /* signifies if user is viewing on screen or printer */
int open_result; /* indicates if error occurred during file opening */
char line_from_file[100];   /* string to read a line of data from file */
int line_count = 0; /* used to determine when screen is full of data */
/* Local functions */
int GetDataLineFromFile(FILE *, char [100]);

   _clearscreen(_GCLEARSCREEN);
   /* Ask if want to output to screen or printer */
   while(TRUE){
      printf("\n\n\n\n\n\tDo you want output to go to the screen or printer?");
      printf("\n\t\t(s) screen; (p) printer; (x) exit:  ");
      choice = getche();
      if((choice == 's') || (choice == 'S')){
         view_out_fp = stdout;
         break;}
      if((choice == 'p') || (choice == 'P')){
         view_out_fp = stdprn;
         screen_view = FALSE;
         break;}
      if((choice == 'x') || (choice == 'X'))
         return;}
   /* Open file or quit if file does not exist */
   view_in_fp = OpenFile(&open_result, file_name_to_view, READ);
   if(open_result == ERROR){
      printf("\n\nFile does not exist!\n");
      printf("\n(press any key)");
      getch();
      return;}
   _clearscreen(_GCLEARSCREEN);
   if(screen_view == FALSE)
      printf("\n\n\tNow printing file...");
   /* Get a line of data from the file or quit if at end of file */
   while(GetDataLineFromFile(view_in_fp, line_from_file) != FILE_END){
      if(screen_view){
         /* Check line counter; stop scrolling if screen is full */
         if(line_count >= FULLSCREEN){
            printf("\t(press 'q' to quit, any other key to continue)\n");
            choice = getch();
            if( (choice ==  'q') || (choice == 'Q') )
               break;
```

```
            line_count = 0;}}
         /* Print line of data to output file */
         fprintf(view_out_fp, "%s", line_from_file);
         if(screen_view)
            line_count += 1;
         else
            fprintf(view_out_fp, "\r");}
      fflush(view_out_fp);
      fclose(view_in_fp);
      printf("\n\n(press any key)");
      getch();
      return;
   }  /* end ShowFile */


void ShowValueChoices(struct formatted_day *formatted_ptr, int month, int day,
        int year)
/*-------------------------------------------------------------------------*/
/* Displays values for a day of data with corresponding numbers so user    */
/*  may choose which value to edit.                                        */
/* Receives: *formatted_ptr:  address of the formatted data                */
/*           month:  month number (1-12)                                   */
/*           day:  day of month                                            */
/*           year:  two-digit year                                         */
/* Globals: value_code                                                     */
/*          units_code                                                     */
/*          month_name                                                     */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* Print the day chosen */
   printf("\n\t\tData for %s %d, %d", month_name[month], day, year);
   /* Print numbers corresponding to values and the value descriptions */
   printf("\n\t\tCHOOSE A VALUE TO CHANGE\n\n");
   printf("\t1.  %6.2f %s  %s\n", formatted_ptr->fmaxt, units_code[CLCIS],
          value_code[VALDMXT]);
   printf("\t2.  %6.2f %s  %s\n", formatted_ptr->fmint, units_code[CLCIS],
          value_code[VALDMNT]);
   printf("\t3.  %6.2f %s  %s\n", formatted_ptr->fsr, units_code[MJPDA],
          value_code[VALDTSR]);
   printf("\t4.  %6.2f %s  %s\n\n", formatted_ptr->fprec, units_code[CMETR],
          value_code[VALDTPR]);
   printf("\tQ.  Quit and Save Changes\n");
   printf("\n\n\tYour Choice:  ");
}  /* end ShowValueChoices */


void ZeroQCValues(struct qc *qcptr)
/*-------------------------------------------------------------------------*/
/* Sets all QC values to zero.                                             */
/* Receives: *qcptr:  address of qc values                                 */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int i; /* loop counter */
int j;  /* loop counter */

   for(i=1; i<=MONTHS; i++)
      for(j=0; j<=MAXMIN-1; j++)
         qcptr->temp[i][j] = 0.0;
   for(i=1; i<=MONTHS; i++)
      for(j=0; j<=MAXMIN-1; j++)
         qcptr->sr[i][j] = 0.0;
   for(i=1; i<=MONTHS; i++)
      for(j=0; j<=MAXMIN-1; j++)
         qcptr->prec[i][j] = 0.0;
   for(i=0; i<=MAXMIN-1; i++)
      qcptr->rh[i] = 0.0;
   for(i=0; i<=MAXMIN-1; i++)
      qcptr->volt[i] = 0.0;
```

```
    for(i=0; i<=MAXMIN-1; i++)
       qcptr->mois[i] = 0.0;
    qcptr->maxtempchng = 0.0;
    return;
}  /* end ZeroQCValues */
```

WGENER.C

```
/* Include files */
#include <conio.h>      /* for console functions */
#include <stdio.h>      /* for file functions */
#include <process.h>    /* for spawn function */
#include <graph.h>      /* for clearscreen function */
#include <string.h>     /* for string functions */
#include "whdr.h"       /* defines, structs, functions for this program */
#include "wglobal.h"    /* global variables for this program */


void GetGeneratedData(void)
/*-------------------------------------------------------------------------*/
/* Gets years of generated data and allows the user to choose one of these */
/*  years to put with the current data or to put in a file by itself;      */
/*  the chosen generated data are then placed in the current/generated     */
/*  data file.                                                             */
/* Receives: none.                                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: AskToUseAllGeneratedData                                         */
/*        SetGeneratedStartingDay                                          */
/*        AllGeneratedDataStartDay                                         */
/*        RunGen                                                           */
/*        GetGenDay                                                        */
/*        FormatGenData                                                    */
/*        GetNewInfileData                                                 */
/*        MoveToChosenYear                                                 */
/*        DisplaySumData                                                   */
/*        OpenFile                                                         */
/*        GetFormattedDay                                                  */
/*        PrintFormattedDay                                                */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */
FILE *current_fp; /* pointer to current data file */
FILE *dailysum_fp; /* pointer to WGEN daily summary file */
FILE *yearlysum_fp; /* pointer to WGEN yearly summary file */
FILE *wgeninput_fp; /* pointer to WGEN input parameters file */
FILE *curgen_fp; /* pointer to current/gen'd data file */
int open_result; /* signifies if error occurred during file opening */
int first_gen_day; /* first day of gen'd data to be put in current/gen'd file */
int first_gen_day_year; /* year of the first day of gen'd data to be put in */
                        /*    the current/gen'd file */
int last_gen_day; /* last day of gen'd data to be put in current/gen'd file */
int last_gen_day_year; /* year of the last day of gen'd data to be put in */
                       /*    the current/gen'd file */
char location_id[3]; /* string for location ID (for IBSNAT format) */
char station_id[3]; /* string for station ID (for IBSNAT format) */
float latitude; /* latitude of weather station (for IBSNAT format) */
float longitude; /* longitude of weather station (for IBSNAT format) */
float parfac; /* factor to convert MJ/m2 to PAR (for IBSNAT format) */
float pardat; /* signifyies if PAR is included in file (for IBSNAT format) */
int use_all_generated_data = FALSE; /* signifies if user wants only gen'd */
                                    /*    data in the current/gen'd file */
struct gen_data gendata; /* gen'd data for a day */
struct formatted_day formatted_data; /* daily summary in IBSNAT format */
int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION];
                                    /* array of QC errors and actions */
int generated_year[YRS_GEN_DATA]; /* array of the year numbers of gen'd data */
char chosen_year_string[5]; /* string to get the listed number of the gen'd */
                            /*    year chosen by user */
int chosen_year; /* listed number of the gen'd year chosen by user */
long start_year_pos; /* location in WGEN daily summary file where the chosen */
                     /*    year of gen'd data begins */
int AskToUseAllGeneratedData(void);
void SetGeneratedStartingDay(int, int, int *, int *);
int AllGeneratedDataStartDay(void);
void RunGen(void);
void FormatGenData(struct gen_data *, struct formatted_day *,
```

```
      int [VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION], int);
void GetNewInfileData(void);
void MoveToChosenYear(FILE *, int);
void DisplaySumData(FILE *, int [YRS_GEN_DATA], int, int, int, int);

   /* Ask if want to use current input file for WGEN or to change it or exit */
   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\n\tDo you want to edit the input file used for WGEN? ");
      printf("\n\t\t(y) yes; (n) no; (x) exit:  ");
      choice = getche();
      if((choice == 'n') || (choice == 'N')){
         /* Make sure input file for WGEN does exist */
         wgeninput_fp = OpenFile(&open_result, "wgen.in", READ);
         if(open_result == ERROR)
            GetNewInfileData();
         else
            fclose(wgeninput_fp);
         break;}
      if((choice == 'y') || (choice == 'Y')){
         GetNewInfileData();
         break;}
      else if((choice == 'x') || (choice == 'X'))
         return;}

   /* Open current/generated IBSNAT file to write to */
   curgen_fp = OpenFile(&open_result, info.curgen_file, WRITE);
   /* Open file of current weather data */
   current_fp = OpenFile(&open_result, info.current_file, READ);
   /* If it does not exist, ask if user wants to make full year of */
   /*   generated data or exit */
   if(open_result == ERROR)
      if((use_all_generated_data = AskToUseAllGeneratedData()) == FALSE)
         return;
   /* Skip if user has already chosen to have all gen'd data */
   if(use_all_generated_data == FALSE){
      /* Skip past line of header data in file of current data */
      if((fscanf(current_fp, "%2s%2s %f %f %f %f\n", location_id, station_id,
            &latitude, &longitude, &parfac, &pardat)) == EOF){
         /* If no data in file, ask if want to use all gen'd data */
         if((use_all_generated_data = AskToUseAllGeneratedData()) == FALSE)
            return;}
      /* If header was input OK, write header to cur/gen'd file */
      else fprintf(curgen_fp, "%2s%2s %6.2f %6.2f %5.2f %5.2f\n", location_id,
            station_id, latitude, longitude, parfac, pardat);}
   /* Skip if user has chosen to have all gen'd data */
   if(use_all_generated_data == FALSE){
      /* Read in first line of data from current file */
      if(GetFormattedDay(current_fp, &formatted_data, erroractions) ==
            FILE_END){
         /* If no data in file, ask if want to use all gen'd data */
         if((use_all_generated_data = AskToUseAllGeneratedData()) == FALSE)
            return;}
      /* Write data to IBSNAT file */
      else{
         printf("\n\n\tNow putting current data in IBSNAT file...\n");
         PrintFormattedDay(curgen_fp, &formatted_data, erroractions);
         /* Set the ending day of gen'd data as a year after the first */
         /*    day of current data */
         last_gen_day = formatted_data.fjdate;
         last_gen_day_year = formatted_data.fyear + 1;
         /* Read in a line of data from current file until EOF */
         while(GetFormattedDay(current_fp, &formatted_data, erroractions) !=
               FILE_END){
            /* Write data to IBSNAT file */
            PrintFormattedDay(curgen_fp, &formatted_data, erroractions);}
         fclose(current_fp);
         /* Set the starting day of gen'd data as the day after the last */
         /*    day of current data */
         SetGeneratedStartingDay(formatted_data.fjdate, formatted_data.fyear,
               &first_gen_day, &first_gen_day_year);}}

   if(use_all_generated_data){
```

```
   if((first_gen_day = AllGeneratedDataStartDay()) == RETURN_TO_MENU)
      return;
   first_gen_day_year = info.year;
   last_gen_day = first_gen_day;
   last_gen_day_year = info.year + 1;
   /* Put header in cur/gen'd data file */
   fprintf(curgen_fp, "%2s%2s %6.2f %6.2f %5.2f %5.2f\n", info.location_id,
         info.station_id, info.lat, info.lon, info.parfac, info.pardat);}

/* Display summary data and let user choose year to use or choose to */
/*    generate new data */
while(TRUE){
   /* See if file of daily summaries  exists; use it if it does exist */
   while(TRUE){
      dailysum_fp = OpenFile(&open_result, WGEN_DAILYFILE, READ);
      if(open_result == ERROR)
         /* Generate data if file doesn't exist */
         RunGen();
      else
         break;}
   /* See if file of yearly summaries exists; use it if it does exist */
   while(TRUE){
      yearlysum_fp = OpenFile(&open_result, WGEN_YEARLYFILE, READ);
      if(open_result == ERROR)
         /* Generate data if file doesn't exist */
         RunGen();
      else
         break;}
   /* Display summary data to user */
   DisplaySumData(yearlysum_fp, generated_year, first_gen_day, last_gen_day,
         first_gen_day_year, last_gen_day_year);
   /* Get user's choice of a year's data to use or to generate new data */
   while(TRUE){
      printf("Your choice:  ");
      gets(chosen_year_string);
      if(chosen_year_string[0] != '\0')
         if(sscanf(chosen_year_string, "%d", &chosen_year) != 0)
            if((chosen_year >= 1) && (chosen_year <= YRS_GEN_DATA+1))
               break;}
   if(chosen_year != YRS_GEN_DATA+1){
      /* Get proper year number corresponding to chosen year */
      chosen_year = generated_year[chosen_year-1];
      break;}
   else{
      /* Generate a new set of data */
      fclose(dailysum_fp);
      fclose(yearlysum_fp);
      RunGen();}}

printf("\n\n\tNow putting generated data in current/generated file...\n");
/* Go to chosen year in daily summary file */
MoveToChosenYear(dailysum_fp, chosen_year);
/* Set starting point in file for the chosen year */
start_year_pos = ftell(dailysum_fp);

/* Move in the daily summary file to day generated data will start */
while(gendata.jday != first_gen_day)
   if(GetGenDay(dailysum_fp, &gendata) == FILE_END){
      /* Exit if encounter end of file early */
      printf("\n3.  Error occured while reading generated data file\n");
      exit(GENERATED_FILE_READ_ERROR);}
/* Read data from daily summary file and put in cur/gen'd file until */
/*    reach the last day for gen'd data */
do{
   /* Convert units of generated data to IBSNAT units */
   FormatGenData(&gendata, &formatted_data, erroractions,
         first_gen_day_year);
   /* Put the formatted_ptr data in the cur/gen'd file */
   PrintFormattedDay(curgen_fp, &formatted_data, erroractions);

   /* Get a day of gen'd data */
   if(GetGenDay(dailysum_fp, &gendata) == FILE_END){
      /* Exit if encounter end of file early */
```

```
            printf("\n4.  Error occured while reading generated data file\n");
            exit(GENERATED_FILE_READ_ERROR);}
         /* Check if have extra day for leap year and don't need it or */
         /*    if have come to start of next gen'd year, then move to beginning */
         /*    of year and get first day of data */
         if( ((gendata.jday == 366)&&((first_gen_day_year % 4) != 0)) ||
               (gendata.jday == 1) ){
            fseek(dailysum_fp, start_year_pos, SEEK_SET);
            /* Increment year */
            first_gen_day_year++;
            if(GetGenDay(dailysum_fp, &gendata) == FILE_END){
               /* Exit if encounter end of file early */
               printf("\n5.  Error occured while reading generated data file\n");
               exit(GENERATED_FILE_READ_ERROR);}}}
      while(gendata.jday != last_gen_day);

      fcloseall();
      printf("\n\n\t--- Current/Generated file now contains one year of data ---");
      printf("\n\n\t\t\t(press any key)");
      getch();
      return;
   }  /* end GetGeneratedData */



   /* Other functions */


   int AllGeneratedDataStartDay(void)
   /*-------------------------------------------------------------------------*/
   /* Asks user for a starting date for generated data in the cur/gen'd file. */
   /* Receives: none.                                                         */
   /* Globals: none.                                                          */
   /* Returns: Julian day to start gen'd data.                                */
   /* Calls: JulDate                                                          */
   /*-------------------------------------------------------------------------*/
   {
   int month; /* month number (1-12) */
   int day; /* day of month */
   char month_day[8]; /* month and day entered by user */
   int first_gen_day; /* first day of gen'd data to be put in current/gen'd file */

      /* Get starting date for generated data */
      while(TRUE){
         _clearscreen(_GCLEARSCREEN);
         printf("\n\n\t(Press 'enter' to exit)\n\n");
         printf("\tEnter the month/day to begin generated data (mm/dd):  ");
         cgets(month_day);
         /* See if just a return was entered */
         if(month_day[1] != 0){
            /* If something is entered, see if it is a date */
            if(sscanf(&month_day[2], "%d/%d", &month, &day) == 2){
               first_gen_day = JulDate(month, day, info.year);
               if((first_gen_day != JD_INVALID_MONTH) && (first_gen_day !=
                  JD_INVALID_DAY))
                  break;}}
         else
            /* Exit if a return was entered */
            return(RETURN_TO_MENU);}
      return(first_gen_day);
   } /* end AllGeneratedDataStartDay */


   int AskToUseAllGeneratedData(void)
   /*-------------------------------------------------------------------------*/
   /* Asks user if a file of only generated data should be created.           */
   /* Receives: none.                                                         */
   /* Globals: none.                                                          */
   /* Returns: TRUE:  if file of gen'd data is to be created                  */
   /*          FALSE:  if user wants to return to main menu                   */
   /* Calls: none.                                                            */
   /*-------------------------------------------------------------------------*/
```

```
{
int choice; /* user's choice */

   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\n\tFile of current weather data does not exist.");
      printf("\n\n\tDo you wish to create a file completely of generated");
      printf("\n   data or exit to the main menu?");
      printf("\n\t\t(c) create; (x) exit:  ");
      choice = getche();
      if((choice == 'c') || (choice == 'C'))
         return(TRUE);
      if((choice == 'x') || (choice == 'X'))
         return(FALSE);}
} /* end AskToUseAllGeneratedData */


void FormatGenData(struct gen_data *gendata_ptr,
      struct formatted_day *formatted_ptr,
      int erroractions[VALUES_WITH_ERRORS][ONE_ERROR_ONE_ACTION], int year)
/*-------------------------------------------------------------------------*/
/* Converts a daily summary in WGEN format to IBSNAT format.               */
/* Receives: *gendata_ptr:  address of a day of gen'd data                 */
/*           *formatted_ptr:  address of the formatted data                */
/*           erroractions[][]:  array of QC errors and actions             */
/*           year:  two-digit year                                         */
/* Globals: info                                                           */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int i; /* loop counter */
int j; /* loop counter */

   strcpy(formatted_ptr->location_id, info.location_id);
   strcpy(formatted_ptr->station_id, info.station_id);
   formatted_ptr->fyear = year;
   formatted_ptr->fjdate = gendata_ptr->jday;
   /* Convert langleys to MJ/m2/day */
   formatted_ptr->fsr = gendata_ptr->sr * 0.0419;
   /* Convert degrees F to degrees C for max and min temps ((F-32)*5/9) */
   formatted_ptr->fmaxt = (gendata_ptr->maxt - FTOC32) * FTOC59;
   formatted_ptr->fmint = (gendata_ptr->mint - FTOC32) * FTOC59;
   /* Convert inches to cm for precip */
   formatted_ptr->fprec = gendata_ptr->prec * INTOCM;
   /* Set array of errors/actions to indicate generated data */
   for(i=0; i<=VALUES_WITH_ERRORS-1; i++)
      for(j=0; j<=ONE_ERROR_ONE_ACTION-1; j++)
         if((i == 0) && (j == 0))
            erroractions[i][j] = 'G';
         else
            erroractions[i][j] = 0;
}  /* end FormatGenData */


void DisplaySumData(FILE *yearlysum_fp, int generated_year[YRS_GEN_DATA],
      int first_gen_day, int last_gen_day, int first_gen_day_year,
      int last_gen_day_year)
/*-------------------------------------------------------------------------*/
/* For each year of gen'd data, a summary is displayed to the user of the  */
/*  ave. temp. and total precip for the period over which gen'd data is    */
/*  required; the summaries are ranked by temp or precip (user's choice).  */
/* Receives: yearlysum_fp:  file pointer to WGEN yearly summary file       */
/*           generated_year[]:  array of the year numbers of the gen'd data*/
/*           first_gen_day:  first day of gen'd data                       */
/*           last_gen_day:  last day of gen'd data                         */
/*           first_gen_day_year:  year of first day of gen'd data          */
/*           last_gen_day_year:  year of last day of gen'd data            */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: GetSumPrecTemp                                                   */
/*        Sort                                                             */
/*-------------------------------------------------------------------------*/
```

```
{
int choice; /* user's choice */
int i; /* loop counter */
float gen_precip[YRS_GEN_DATA]; /* total precip for each gen'd year */
float gen_temp[YRS_GEN_DATA]; /* ave. temp for each gen'd year */

void GetSumPrecTemp(FILE *, float *, float *, int, int, int, int);
void Sort(int, float [], float [], int []);

   /* Store year, precip and ave of max/min temp for each year in array */
   for(i=0; i<YRS_GEN_DATA; i++){
      GetSumPrecTemp(yearlysum_fp, &gen_precip[i], &gen_temp[i], first_gen_day,
            last_gen_day, first_gen_day_year, last_gen_day_year);
      generated_year[i] = i + 1;}
   /* Get value to sort by from user: precip or temp */
   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\n\tYou will be shown summaries for the remainder of the\n");
      printf("\tseason for %d different years of generated data.\n",
            YRS_GEN_DATA);
      printf("\tChoose the summary which best reflects the type of data\n");
      printf("\tyou wish to add to the current/generated file.\n");
      printf("\n\tDo you want the summaries ranked by precip. or temperature?");
      printf("\n\t\t(p) precip.; (t) temperature:  ");
      choice = getche();
      /* Sort according to the value chosen by user */
      if((choice == 'p') || (choice == 'P')){
         choice = 'p';
         Sort(YRS_GEN_DATA, gen_precip, gen_temp, generated_year);
         break;}
      if((choice == 't') || (choice == 'T')){
         choice = 't';
         Sort(YRS_GEN_DATA, gen_temp, gen_precip, generated_year);
         break;}}

   /* Display the data */
   _clearscreen(_GCLEARSCREEN);
   for(i=0; i<YRS_GEN_DATA; i++){
      if(choice == 'p'){
         if(i==0){
            printf("Summaries for the remainder of the season, ranked by\n");
            printf("total precip. (ave. temperature in brackets)\n\n");}
         printf("\t%2d.   ", i+1);
         printf("%5.1f inches     [%5.1f F]\n", gen_precip[i], gen_temp[i]);}
      else{
         if(i==0){
            printf("Summaries for the remainder of the season, ranked by\n");
            printf("ave. temperature (total precip. in brackets)\n\n");}
         printf("\t%2d.   ", i+1);
         printf("%5.1f F     [%5.1f inches]\n", gen_temp[i], gen_precip[i]);}}
   printf("\t%2d.   Generate new data\n\n", YRS_GEN_DATA+1);
   return;
}  /* end DisplaySumData */


void GetInfileValue(float *wgen_input_value)
/*-------------------------------------------------------------------------*/
/* Gets a value for the WGEN input file from user.                         */
/* Receives: *wgen_input_value:  address of value for WGEN input file      */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
char wgen_input_string[12]; /* used to get wgen input value entered by user */
float entered_value; /* numeric value from the string entered by user */

   /* Repeat until get a valid answer */
   while(TRUE){
      /* Get input from user */
      gets(wgen_input_string);
      /* If just a return entered, don't change value and exit */
      if(wgen_input_string[0] == '\0')
```

```
         return;
      /* If something is entered, see if it is a number */
      else
         /* If a number entered, set value equal to the number and exit */
         if(sscanf(wgen_input_string, "%f", &entered_value) != 0){
            *wgen_input_value = entered_value;
            return;}}
} /* end GetInfileValue */


void GetNewInfileData(void)
/*-------------------------------------------------------------------------*/
/* Gets input parameters to run WGEN from user.                            */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: GetInfileValue                                                   */
/*        LoadInfileData                                                   */
/*        InitializeInfileValues                                           */
/*        SaveInfileData                                                   */
/*-------------------------------------------------------------------------*/
{
int open_result; /* signifies if error occurred during file opening */
int finished = FALSE; /* signifies when user is finished editing */
char wgen_input_desc[81]; /* string to get comment line for WGEN input file */
                          /*    from user */
int choice; /* user's choice */
int i; /* loop counter */
struct wgen_input wgen_input_data; /* data for the WGEN input file */
void GetInfileValue(float *);
int LoadInfileData(struct wgen_input*);
void InitializeInfileValues(struct wgen_input*);
void SaveInfileData(struct wgen_input *);

   /* Read in old data to use as defaults, if available */
   if(LoadInfileData(&wgen_input_data) != OK)
      /* Set values to "zero"s if no old data */
      InitializeInfileValues(&wgen_input_data);
   while(finished == FALSE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\tEnter the WGEN input file data as you are prompted.\n\n");
      printf("\tExisting values are given in brackets.\n");
      printf("\tPress 'enter' to keep existing value.\n\n");

      printf("\n\nEnter a brief description for the file ");
      printf("(no more than 75 characters):\n");
      printf("%s\n", wgen_input_data.user_comments);
      gets(wgen_input_desc);
      if(wgen_input_desc[0] != '\0')
         sprintf(wgen_input_data.user_comments, "[%s]", wgen_input_desc);

      printf("\nObtain the following values from the proper WGEN\n");
      printf("   tables and charts for the location nearest yours.\n\n");
      for(i=0; i<MONTHS; i++){
         printf("   Enter P(W/W) for %s:  ", month_name[i]);
         printf("[%6.3f]  ", wgen_input_data.probwet_wet[i]);
         GetInfileValue(&wgen_input_data.probwet_wet[i]);}
      for(i=0; i<MONTHS; i++){
         printf("   Enter P(W/D) for %s:  ", month_name[i]);
         printf("[%6.3f]  ", wgen_input_data.probwet_dry[i]);
         GetInfileValue(&wgen_input_data.probwet_dry[i]);}
      for(i=0; i<MONTHS; i++){
         printf("   Enter ALPHA for %s:  ", month_name[i]);
         printf("[%6.3f]  ", wgen_input_data.alpha[i]);
         GetInfileValue(&wgen_input_data.alpha[i]);}
      for(i=0; i<MONTHS; i++){
         printf("   Enter BETA for %s:  ", month_name[i]);
         printf("[%6.3f]  ", wgen_input_data.beta[i]);
         GetInfileValue(&wgen_input_data.beta[i]);}
      for(i=0; i<12; i++){
         switch(i){
            case 0:
               printf("   Enter TXMD:  ");
```

```
                break;
            case 1:
                printf("   Enter ATX:  ");
                break;
            case 2:
                printf("   Enter CVTX:  ");
                break;
            case 3:
                printf("   Enter ACVTX:  ");
                break;
            case 4:
                printf("   Enter TXMW:  ");
                break;
            case 5:
                printf("   Enter TN:  ");
                break;
            case 6:
                printf("   Enter ATN:  ");
                break;
            case 7:
                printf("   Enter CVTN:  ");
                break;
            case 8:
                printf("   Enter ACVTN:  ");
                break;
            case 9:
                printf("   Enter RMD:  ");
                break;
            case 10:
                printf("   Enter AR:  ");
                break;
            case 11:
                printf("   Enter RMW:  ");
                break;}
         printf("[%6.3f]  ", wgen_input_data.wdparams[i]);
         GetInfileValue(&wgen_input_data.wdparams[i]);}
      while(TRUE){
         printf("\nDo you need to reenter any of these values? ");
         choice = getche();
         if((choice == 'n') || (choice == 'N')){
            finished = TRUE;
            break;}
         if((choice == 'y') || (choice == 'Y'))
            break;}}
   SaveInfileData(&wgen_input_data);
   return;
}  /* end GetNewInfileData */


void GetSumPrecTemp(FILE *yearlysum_fp, float *total_gen_precip,
      float *ave_gen_temp, int first_gen_day, int last_gen_day,
      int first_gen_day_year, int last_gen_day_year)
/*-------------------------------------------------------------------------*/
/* For a year of gen'd data, the ave. temp. and total precip. are          */
/*    calculated for the period over which gen'd data is required.         */
/* Receives: yearlysum_fp:  pointer to WGEN yearly summary file            */
/*           *total_gen_precip:  address of total precip                   */
/*           *ave_gen_temp:  address of ave. temp.                         */
/*           first_gen_day:  first day of gen'd data                       */
/*           last_gen_day:  last day of gen'd data                         */
/*           first_gen_day_year:  year of first day of gen'd data          */
/*           last_gen_day_year:  year of last day of gen'd data            */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: CalDate                                                          */
/*-------------------------------------------------------------------------*/
{
int last_gen_day_month; /* month of last day of gen'd data */
int first_gen_day_month; /* month of first day of gen'd data */
int day; /* day of month */
int divisor_for_averages = 0; /* number of months used to calculate ave. temp */
char line_from_file[81]; /* string used to read a line of data from file */
float yearlysum_precip; /* precip value read from WGEN yearly summary file */
```

```
float gen_maxt_total = 0.0; /* used to calculate ave. temp. */
float gen_mint_total = 0.0; /* used to calculate ave. temp. */
float yearlysum_maxt; /* max temp. value read from WGEN yearly summary file */
float yearlysum_mint; /* min temp. value read from WGEN yearly summary file */
int i; /* loop counter */

   *total_gen_precip = 0.0;
   /* Get month that the first and last days of gen'd data are in */
   CalDate(first_gen_day, first_gen_day_year, &first_gen_day_month, &day);
   CalDate(last_gen_day, last_gen_day_year, &last_gen_day_month, &day);

   /* Skip past header, and wet days, and rainfall heading */
   for(i=1; i<=6; i++)
      if(fgets(line_from_file, 80, yearlysum_fp) == NULL){
         printf("\nError occured while reading generated data summary file\n");
         exit(SUMMARY_FILE_READ_ERROR);}
   /* Read in rainfall values and add to total if they are in */
   /*    the range that is going to be used */
   for(i=1; i<=12; i++){
      if(fscanf(yearlysum_fp, " %f ", &yearlysum_precip) == EOF){
         printf("\nError occured while reading generated data summary file\n");
         exit(SUMMARY_FILE_READ_ERROR);}
      /* If first and last months are in different years, see if this monthly */
      /*    total is after or equal to the first month or before or equal to */
      /*    the last month */
      if(first_gen_day_month >= last_gen_day_month){
         if((i >= first_gen_day_month) || (i <= last_gen_day_month))
            *total_gen_precip += yearlysum_precip;}
      else
         /* If first and last months are in same year, see if this monthly */
         /*    total is after or equal to the first month and before or equal */
         /*    to the last month */
         if((i >= first_gen_day_month) && (i <= last_gen_day_month))
            *total_gen_precip += yearlysum_precip;}
   /* Skip over year total */
   if(fscanf(yearlysum_fp, " %f ", &yearlysum_precip) == EOF){
      printf("\nError occured while reading generated data summary file\n");
      exit(SUMMARY_FILE_READ_ERROR);}
   /* Skip over max temp heading */
   if(fgets(line_from_file, 80, yearlysum_fp) == NULL){
         printf("\nError occured while reading generated data summary file\n");
         exit(SUMMARY_FILE_READ_ERROR);}
   /* Read in max temperature values; add to total if they are in */
   /*    the range that is going to be used and increment counter for ave */
   for(i=1; i<=12; i++){
      if(fscanf(yearlysum_fp, " %f ", &yearlysum_maxt) == EOF){
         printf("\nError occured while reading generated data summary file\n");
         exit(SUMMARY_FILE_READ_ERROR);}
      if(first_gen_day_month >= last_gen_day_month){
         if((i >= first_gen_day_month) || (i <= last_gen_day_month))
            gen_maxt_total += yearlysum_maxt;
            divisor_for_averages++;}
      else
         if((i >= first_gen_day_month) && (i <= last_gen_day_month)){
            gen_maxt_total += yearlysum_maxt;
            divisor_for_averages++;}}
   /* Skip over year ave */
   if(fscanf(yearlysum_fp, " %f ", &yearlysum_maxt) == EOF){
      printf("\nError occured while reading generated data summary file\n");
      exit(SUMMARY_FILE_READ_ERROR);}
   /* Skip over min temp header */
   if(fgets(line_from_file, 80, yearlysum_fp) == NULL){
         printf("\nError occured while reading generated data summary file\n");
         exit(SUMMARY_FILE_READ_ERROR);}
   /* Read in min temperature values; add to total if they are in */
   /*    the range that is going to be used */
   for(i=1; i<=12; i++){
      if(fscanf(yearlysum_fp, " %f ", &yearlysum_mint) == EOF){
         printf("\nError occured while reading generated data summary file\n");
         exit(SUMMARY_FILE_READ_ERROR);}
      if(first_gen_day_month >= last_gen_day_month){
         if((i >= first_gen_day_month) || (i <= last_gen_day_month))
            gen_mint_total += yearlysum_mint;}
```

```
      else
         if((i >= first_gen_day_month) && (i <= last_gen_day_month))
            gen_mint_total += yearlysum_mint;}
   /* Skip over year ave */
   if(fscanf(yearlysum_fp, " %f ", &yearlysum_mint) == EOF){
      printf("\nError occured while reading generated data summary file\n");
      exit(SUMMARY_FILE_READ_ERROR);}
   for(i=1; i<=3; i++)
      if(fgets(line_from_file, 80, yearlysum_fp) == NULL){
         printf("\nError occured while reading generated data summary file\n");
         exit(SUMMARY_FILE_READ_ERROR);}
   *ave_gen_temp = ((gen_maxt_total/(float) divisor_for_averages) +
         (gen_mint_total/(float) divisor_for_averages)) / 2.0;
   return;
} /* end GetSumPrecTemp */


void InitializeInfileValues(struct wgen_input *wgen_input_ptr)
/*-------------------------------------------------------------------------*/
/* Initializes WGEN input parameters for editing.                          */
/* Receives: *wgen_input_ptr:  address of data for WGEN input file         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int i; /* loop counter */

   strcpy(wgen_input_ptr->user_comments, "[WGEN input file]");
   wgen_input_ptr->n_years = YRS_GEN_DATA;
   wgen_input_ptr->gen_code = 1;
   wgen_input_ptr->lat_deg = info.lat;
   wgen_input_ptr->tcf_code = 0;
   wgen_input_ptr->rcf_code = 0;
   for(i=0; i<12; i++)
      wgen_input_ptr->probwet_wet[i] = 0.0;
   for(i=0; i<12; i++)
      wgen_input_ptr->probwet_dry[i] = 0.0;
   for(i=0; i<12; i++)
      wgen_input_ptr->alpha[i] = 0.0;
   for(i=0; i<12; i++)
      wgen_input_ptr->beta[i] = 0.0;
   for(i=0; i<12; i++)
      wgen_input_ptr->wdparams[i] = 0.0;
   return;
} /* end InitializeInfileValues */


int LoadInfileData(struct wgen_input *wgen_input_ptr)
/*-------------------------------------------------------------------------*/
/* Loads WGEN input parameters for editing.                                */
/* Receives: *wgen_input_ptr:  address of data for WGEN input file         */
/* Globals: none.                                                          */
/* Returns: OK:  if info successfully loaded                               */
/*          ERROR:  if some error occured                                  */
/* Calls: ScanInfileData                                                   */
/*-------------------------------------------------------------------------*/
{
FILE *wgeninput_fp;  /* pointer for WGEN input file */
int open_result;   /* signifies if error occured when opening file */
int ScanInfileData(FILE *, struct wgen_input *);

   /* Open file and check if error occured (file doesn't exist) */
   wgeninput_fp = OpenFile(&open_result, "wgen.in", READ);
   if(open_result == ERROR)
      return(ERROR);
   /* Load data into struct; stop if insufficient data in file */
   if(ScanInfileData(wgeninput_fp, wgen_input_ptr) == FILE_END){
      fclose(wgeninput_fp);
      return(ERROR);}
   fclose(wgeninput_fp);
   /* Return OK to show data is loaded */
   return(OK);
```

```
} /* end LoadInfileData */


void MoveToChosenYear(FILE *dailysum_fp, int gen_year_number)
/*-------------------------------------------------------------------------*/
/* Moves a file pointer to the beginning of the chosen year of gen'd data  */
/*  in the WGEN daily summary file.                                        */
/* Receives: *dailysum_fp:  file pointer to the WGEN daily summary file    */
/*           gen_year_number:  number of the year of gen'd data chosen     */
/*                             by user                                     */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: GetGenDay                                                        */
/*-------------------------------------------------------------------------*/
{
int gen_year_end; /* number of days in the year of gen'd data chosen by user */
struct gen_data gendata; /* gen'd data for a day */

   /* If using first year's data, just return */
   if(gen_year_number == 1)
      return;
   /* Move to the beginning of the year before the desired year */
   do{
      if(GetGenDay(dailysum_fp, &gendata) == FILE_END){
         /* Exit if encounter end of file early */
         printf("\n1.  Error occured while reading generated data file\n");
         exit(GENERATED_FILE_READ_ERROR);}}
   while(gendata.year != gen_year_number-1);
   /* Determine the length of the year before the desired year */
   if( ((gen_year_number-1) %4) == 0)
      gen_year_end = 366;
   else
      gen_year_end = 365;
   /* Move to the beginning of the desired year */
   do{
      if(GetGenDay(dailysum_fp, &gendata) == FILE_END){
         /* Exit if encounter end of file early */
         printf("\n2.  Error occured while reading generated data file\n");
         exit(GENERATED_FILE_READ_ERROR);}}
      while(gendata.jday != gen_year_end);
} /* end MoveToChosenYear */


void RunGen(void)
/*-------------------------------------------------------------------------*/
/* Runs WGEN to produce a daily summary file and a yearly summary file.    */
/* Receives: none.                                                         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int rungen_result; /* signifies if an error occurred while running WGEN */

   printf("\n\nNow running WGEN to get generated data...\n\n\n");
   rungen_result = spawnl(P_WAIT, "wgenpc", "wgenpc", NULL);
   if(rungen_result != 0){
      printf("\nError occured while running WGEN\n");
      exit(WGEN_ERROR);}
   return;
} /* end RunGen */


void SaveInfileData(struct wgen_input *wgen_input_ptr)
/*-------------------------------------------------------------------------*/
/* Saves WGEN input parameters in a file.                                  */
/* Receives: *wgen_input_ptr:  address of data for WGEN input file         */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: OpenFile                                                         */
/*-------------------------------------------------------------------------*/
{
int i;   /* loop counter  */
```

```
FILE *wgeninput_fp; /* pointer to WGEN input file */
int open_result; /* signifies if error occurred during file opening */
int print_result; /* signifies if error occurred when printing to file */

   /* Open file */
   wgeninput_fp = OpenFile(&open_result, "wgen.in", WRITE);
   if(open_result == ERROR){
      printf("\nError occured while saving WGEN input data\n");
      exit(INPUT_FILE_WRITE_ERROR);}

   /* Print file description */
   if(fputs(wgen_input_ptr->user_comments, wgeninput_fp) == EOF){
      printf("\nError occured while saving WGEN input data\n");
      exit(INPUT_FILE_WRITE_ERROR);}
   fprintf(wgeninput_fp, "\n");
   /* Print years, codes and latitude */
   print_result = fprintf(wgeninput_fp, "%5d%5d%5.1f%5d%5d\n",
         wgen_input_ptr->n_years, wgen_input_ptr->gen_code,
         wgen_input_ptr->lat_deg, wgen_input_ptr->tcf_code,
         wgen_input_ptr->rcf_code);
   if(print_result != 26){
      printf("\nError occured while saving WGEN input data\n");
      exit(INPUT_FILE_WRITE_ERROR);}
   for(i=0; i<12; i++){
      print_result = fprintf(wgeninput_fp, "%6.3f",
            wgen_input_ptr->probwet_wet[i]);
      if(print_result != 6){
         printf("\nError occured while saving WGEN input data\n");
         exit(INPUT_FILE_WRITE_ERROR);}}
   fprintf(wgeninput_fp, "\n");
   for(i=0; i<12; i++){
      print_result = fprintf(wgeninput_fp, "%6.3f",
            wgen_input_ptr->probwet_dry[i]);
      if(print_result != 6){
         printf("\nError occured while saving WGEN input data\n");
         exit(INPUT_FILE_WRITE_ERROR);}}
   fprintf(wgeninput_fp, "\n");
   for(i=0; i<12; i++){
      print_result = fprintf(wgeninput_fp, "%6.3f", wgen_input_ptr->alpha[i]);
      if(print_result != 6){
         printf("\nError occured while saving WGEN input data\n");
         exit(INPUT_FILE_WRITE_ERROR);}}
   fprintf(wgeninput_fp, "\n");
   for(i=0; i<12; i++){
      print_result = fprintf(wgeninput_fp, "%6.3f", wgen_input_ptr->beta[i]);
      if(print_result != 6){
         printf("\nError occured while saving WGEN input data\n");
         exit(INPUT_FILE_WRITE_ERROR);}}
   fprintf(wgeninput_fp, "\n");
   for(i=0; i<4; i++){
      print_result = fprintf(wgeninput_fp, "%8.2f",
            wgen_input_ptr->wdparams[i]);
      if(print_result != 8){
         printf("\nError occured while saving WGEN input data\n");
         exit(INPUT_FILE_WRITE_ERROR);}}
   print_result = fprintf(wgeninput_fp, "\n%8.2f\n",
         wgen_input_ptr->wdparams[4]);
   if(print_result != 10){
      printf("\nError occured while saving WGEN input data\n");
      exit(INPUT_FILE_WRITE_ERROR);}
   for(i=5; i<9; i++){
      print_result = fprintf(wgeninput_fp, "%8.2f",
            wgen_input_ptr->wdparams[i]);
      if(print_result != 8){
         printf("\nError occured while saving WGEN input data\n");
         exit(INPUT_FILE_WRITE_ERROR);}}
   print_result = fprintf(wgeninput_fp, "\n%8.2f%8.2f\n%8.2f\n",
         wgen_input_ptr->wdparams[9], wgen_input_ptr->wdparams[10],
         wgen_input_ptr->wdparams[11]);
   if(print_result != 27){
      printf("\nError occured while saving WGEN input data\n");
      exit(INPUT_FILE_WRITE_ERROR);}
   fclose(wgeninput_fp);
```

```
   return;
}  /* end SaveInfileData */


int ScanInfileData(FILE *wgeninput_fp, struct wgen_input *wgen_input_ptr)
/*-------------------------------------------------------------------------*/
/* Gets data from WGEN input file and puts it into wgen_input struct.      */
/* Receives:  *wgeninput_fp:  pointer to WGEN input file                   */
/*            *wgen_input_ptr:  address of data for WGEN input file        */
/* Returns:  OK:  if data successfully loaded                              */
/*           FILE_END:  if inadequate data in file                         */
/* Calls:  none                                                            */
/*-------------------------------------------------------------------------*/
{
int i;   /* loop counter */

   /* Get file description */
   if(fgets(wgen_input_ptr->user_comments, 80, wgeninput_fp) == NULL)
      /* If at end of file already, return error */
      return(FILE_END);
   /* Get years, codes and latitude */
   if(fscanf(wgeninput_fp, " %d %d %f %d %d ", &wgen_input_ptr->n_years,
         &wgen_input_ptr->gen_code, &wgen_input_ptr->lat_deg,
         &wgen_input_ptr->tcf_code, &wgen_input_ptr->rcf_code) == EOF)
      /* If at end of file already, return error */
      return(FILE_END);
   for(i=0; i<12; i++)
      if(fscanf(wgeninput_fp, " %f ", &wgen_input_ptr->probwet_wet[i]) == EOF)
         /* If at end of file already, return error */
         return(FILE_END);
   for(i=0; i<12; i++)
      if(fscanf(wgeninput_fp, " %f ", &wgen_input_ptr->probwet_dry[i]) == EOF)
         /* If at end of file already, return error */
         return(FILE_END);
   for(i=0; i<12; i++)
      if(fscanf(wgeninput_fp, " %f ", &wgen_input_ptr->alpha[i]) == EOF)
         /* If at end of file already, return error */
         return(FILE_END);
   for(i=0; i<12; i++)
      if(fscanf(wgeninput_fp, " %f ", &wgen_input_ptr->beta[i]) == EOF)
         /* If at end of file already, return error */
         return(FILE_END);
   for(i=0; i<12; i++)
      if(fscanf(wgeninput_fp, " %f ", &wgen_input_ptr->wdparams[i]) == EOF)
         /* If at end of file already, return error */
         return(FILE_END);
   return(OK);
}  /* end ScanInfileData */


void SetGeneratedStartingDay(int last_current_day, int last_current_day_year,
      int *first_gen_day, int *first_gen_day_year)
/*-------------------------------------------------------------------------*/
/* Figures the starting day for generated data as the day after current    */
/*    data ends.                                                           */
/* Receives: last_current_day:  last day of current data                   */
/*           last_current_day_year:  year of last day of current data      */
/*           *first_gen_day:  value of first day of gen'd data to be placed*/
/*              in this address when calculated                            */
/*           *first_gen_day_year:  value of year of first day of gen'd data*/
/*              to be placed in this address when calculated               */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   /* If a leap year... */
   if((last_current_day_year % 4) == 0){
      /* If last day of year... */
      if(last_current_day == 366){
         /* Set the first gen'd day and year */
         *first_gen_day = 1;
         *first_gen_day_year = last_current_day_year + 1;}
```

```
      /* If not last day of year... */
      else{
         /* Set the first gen'd day and year */
         *first_gen_day = last_current_day + 1;
         *first_gen_day_year = last_current_day_year;}}
   /* If not a leap year... */
   else{
      /* If last day of year... */
      if(last_current_day == 365){
         /* Set the first gen'd day and year */
         *first_gen_day = 1;
         *first_gen_day_year = last_current_day_year + 1;}
      /* If not the last day of year... */
      else{
         /* Set the first gen'd day and year */
         *first_gen_day = last_current_day + 1;
         *first_gen_day_year = last_current_day_year;}}
} /* end SetGeneratedStartingDay */


void Sort(int number_to_sort, float first_array[], float second_array[],
      int third_array[])
/*-------------------------------------------------------------------------*/
/* Insertion sort function; sorts on key array and makes the same changes  */
/*  to the two other arrays to keep corresponding data together.           */
/* Receives: number_to_sort:  number of values to sort                     */
/*           first_array:  array of precip or temp summaries to sort by    */
/*           second_array:  follows sorting of first array                 */
/*           third_array:  array of year numbers; follows first array      */
/* Globals: none.                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int i; /* loop counter */
int j; /* loop counter */
float temporary1; /* holds value being sorted */
float temporary2; /* holds value being sorted */
int temporary3; /* holds value being sorted */

   for(i=1; i<number_to_sort; i++){
      temporary1 = first_array[i];
      temporary2 = second_array[i];
      temporary3 = third_array[i];
      j = i-1;
      while ((j > -1) && (first_array[j] > temporary1)){
         first_array[j+1] = first_array[j];
         second_array[j+1] = second_array[j];
         third_array[j+1] = third_array[j];
         j--;}
      first_array[j+1] = temporary1;
      second_array[j+1] = temporary2;
      third_array[j+1] = temporary3;}
   return;
} /* Sort */
```

WMISC.C

```
/* Include files */
#include <conio.h>      /* for console functions */
#include <stdio.h>      /* for file functions */
#include <graph.h>      /* for clearscreen function */
#include <string.h>     /* for string functions */
#include <process.h>    /* for spawn function */
#include <stdlib.h>     /* for int to string conversion */
#include "whdr.h"       /* defines, structs, functions for this program */
#include "wglobal.h"    /* global variables for this program */


void MiscFunctions()
/*-------------------------------------------------------------------------*/
/* Lets user monitor the weather station and set the datalogger's clock;   */
/*  edit some of the program information; or start a new year of data      */
/*  collection.                                                            */
/* Receives: none.                                                         */
/* Globals: none                                                           */
/* Returns: none.                                                          */
/* Calls: ShowMiscMenu                                                     */
/*        MonitorStation                                                   */
/*        EditProgramInfo                                                  */
/*        StartNewDataYear                                                 */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */
void ShowMiscMenu(void);
void MonitorStation(void);
void StartNewDataYear(void);

   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      ShowMiscMenu();
      choice = getche();
      if(choice == '4')
         break;
      switch(choice){
         case '1':
            MonitorStation();
            break;
         case '2':
            EditProgramInfo(EDITING_INFO);
            break;
         case '3':
            StartNewDataYear();
            break;
         default:
            printf("\n\n*** Enter 1-4 ***\n");
            getch();}}
}  /* end MiscFunctions */


void ShowMiscMenu()
/*-------------------------------------------------------------------------*/
/* Prints misc. functions menu to screen.                                  */
/* Receives: none.                                                         */
/* Globals:  none                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   printf("\n\n\n\t\tMENU FOR MISCELLANEOUS FUNCTIONS\n\n\n");
   printf("\t\t1.  Monitor Weather Station, Set Station Date\n\n");
   printf("\t\t2.  Edit Program Information\n\n");
   printf("\t\t3.  Start New Year of Data Collection\n\n");
   printf("\t\t4.  Return to Main Menu\n\n\n");
   printf("\t\tYour choice:  ");
}  /* end ShowMiscMenu */
```

```
/* Other functions */


void EditProgramInfo(int edit_course)
/*-------------------------------------------------------------------------*/
/* Allows user to edit information used by the program.                    */
/* Receives: edit_course:  signifies course user will follow while         */
/*                         editing program info                            */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: InitializeProgramInfo                                            */
/*        SaveProgramInfo                                                  */
/*        PrintProgramInfo                                                 */
/*        OpenFile                                                         */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */
int finished = FALSE; /* indicates when user is finished editing */
char station_name_line[9]; /* used to read the station name from user */
char file_line[15]; /* used to get file names from user */
char id_line[5]; /* used to get ID's from user */
char latlon_line[9]; /* used to get latitude and longitude from user */
char year_line[5]; /* used to get year data collection started from user */
float newlatlon; /* new value for latitude or longitude */
int year; /* two-digit year */
FILE *curgen_fp; /* pointer to current/gen'd file */
int open_result; /* signifies if error occurred during file opening */
char year_string[3]; /* year data collection starts; used in file names */
char old_QC_file[13]; /* name of old QC values file; used to rename file */

void InitializeProgramInfo(void);
void PrintProgramInfo(void);

   station_name_line[0] = 7;
   file_line[0] = 13;
   id_line[0] = 3;
   latlon_line[0] = 7;
   year_line[0] = 3;

   /* If no file of info exists, user must edit info */
   if(edit_course == NO_INFO_FILE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\t\tFile of program info does not exist!\n\n\n");
      printf("\tAfter completing the creation of the program info file, make");
      printf("\n\t   sure you create a file of QC values by moving to the");
      printf("\n\t   VIEW/EDIT MENU and choosing Edit QC Values.\n\n");
      InitializeProgramInfo();}
   else if(edit_course == NEW_DATA_YEAR){
      printf("\n\tThe file of existing current data has been renamed with the");
      printf("\n\t   name previously given to the current/generated data file.");
      printf("\n\t   You should enter a different name for the new current/");
      printf("\n\t   generated file.");
      printf("\n\tThe existing file of QC values will be renamed to its new");
      printf("\n\t   name so these values will not need to be re-entered.");
      printf("\n\n");
      strcpy(old_QC_file, info.qc_file);}
   else if(edit_course == EDITING_INFO){
      /* Ask if want to proceed with edit */
      while(TRUE){
         _clearscreen(_GCLEARSCREEN);
         printf("\n\tYou may only change the latitude and longitude values");
         printf("\n\t   and the information in the datalogger communications");
         printf("\n\t   file at this time.  To change the other information,");
         printf("\n\t   you must choose the Start New Year of Data Collection");
         printf("\n\t   option.");
         printf("\n\tDo you want to proceed with the edit? ");
         choice = getche();
         if((choice == 'y') || (choice == 'Y')){
            printf("\n\n");
            break;}
         if((choice == 'n') || (choice == 'N'))
            return;}
         }
```

```
while(finished == FALSE){
   if(edit_course != EDITING_INFO){
      printf("\n\tEnter the information as you are prompted.\n");
      printf("\tExisting values are given.\n");
      printf("\tPress 'enter' to keep existing value.\n\n");

      printf("The name of the weather station may contain up to six letters\n");
      printf("   and numbers.  The year data collection starts is added to\n");
      printf("   this name to identify some program files.\n");
      printf("Existing:  %s     ", info.sta_name);
      printf("Enter the weather station name:  ");
      cgets(station_name_line);
      if(station_name_line[1] != 0)
         strcpy(info.sta_name, &station_name_line[2]);

      while(TRUE){
         printf("\n\nThe name for the IBSNAT file which will contain ");
         printf("current\n   and generated data must be a legal DOS file ");
         printf("name (xxxxxxxx.xxx).\n");
         printf("Existing:  %s    \n", info.curgen_file);
         printf("   Enter the current/generated data file name:  ");
         cgets(file_line);
         if(file_line[1] != 0)
            strcpy(info.curgen_file, &file_line[2]);
         /* Check to see if this file exists */
         curgen_fp = OpenFile(&open_result, info.curgen_file, READ);
         fclose(curgen_fp);
         /* If file does exists, tell user and get another name */
         if(open_result == OK)
            printf("\n\nThis file already exists!\nEnter a new name.");
         else
            break;}

      printf("\n\nThe location ID must be two letters or numbers.\n");
      printf("Existing:  %s     ", info.location_id);
      printf("Enter the location ID:  ");
      cgets(id_line);
      if(id_line[1] != 0)
         strcpy(info.location_id, &id_line[2]);
      printf("\n\nThe weather station ID must be two letters or numbers.\n");
      printf("Existing:  %s     ", info.station_id);
      printf("Enter the station ID:  ");
      cgets(id_line);
      if(id_line[1] != 0)
         strcpy(info.station_id, &id_line[2]);}

   printf("\n\n");
   while(TRUE){
      printf("Existing:  %6.2f     ", info.lat);
      printf("Enter your latitude:  ");
      cgets(latlon_line);
      /* See if just a return was entered */
      if(latlon_line[1] != 0){
         /* If something is entered, see if it is a number */
         if(sscanf(&latlon_line[2], "%f", &newlatlon) == 1){
            info.lat = newlatlon;
            break;}}
      else
         break;}
   printf("\n\n");
   while(TRUE){
      printf("Existing:  %6.2f     ", info.lon);
      printf("Enter your longitude:  ");
      cgets(latlon_line);
      /* See if just a return was entered */
      if(latlon_line[1] != 0){
         /* If something is entered, see if it is a number */
         if(sscanf(&latlon_line[2], "%f", &newlatlon) == 1){
            info.lon = newlatlon;
            break;}}
      else
         break;}
```

```
   if(edit_course != EDITING_INFO){
      while(TRUE){
         printf("\n\nThe current year must be entered as");
         printf(" the last two digits of the year.\n");
         printf("Existing:  %d     ", info.year);
         printf("Enter the current year:  ");
         cgets(year_line);
         /* See if just a return was entered */
         if(year_line[1] != 0){
            /* If something is entered, see if it is a year */
            if(sscanf(&year_line[2], "%d", &year) == 1){
               info.year = year;
               break;}}
         else
            break;}

      itoa(info.year, year_string, 10);

      strcpy(info.raw_file, info.sta_name);
      strcpy(info.prev_file, info.sta_name);
      strcpy(info.backup_file, info.sta_name);
      strcpy(info.current_file, info.sta_name);
      strcpy(info.qc_file, info.sta_name);

      strcat(info.prev_file, year_string);
      strcat(info.backup_file, year_string);
      strcat(info.current_file, year_string);
      strcat(info.qc_file, year_string);

      strcat(info.raw_file, ".dat");
      strcat(info.prev_file, ".pr");
      strcat(info.backup_file, ".bu");
      strcat(info.current_file, ".cu");
      strcat(info.qc_file, ".qc");

      PrintProgramInfo();}

   while(TRUE){
      printf("\nDo you need to reenter any of this information? ");
      choice = getche();
      if((choice == 'n') || (choice == 'N')){
         finished = TRUE;
         break;}
      if((choice == 'y') || (choice == 'Y')){
         _clearscreen(_GCLEARSCREEN);
         break;}}}

SaveProgramInfo();

if(edit_course == NEW_DATA_YEAR)
   rename(old_QC_file, info.qc_file);

/* See if need to edit weather station communications file */
while(TRUE){
   _clearscreen(_GCLEARSCREEN);
   if(edit_course == NO_INFO_FILE)
      break;
   printf("\n\nDo you need to edit the datalogger communications file? ");
      choice = getche();
      if((choice == 'y') || (choice == 'Y'))
         break;
      if((choice == 'n') || (choice == 'N'))
         return;}

/* Give message to user and call telcom to edit info */
_clearscreen(_GCLEARSCREEN);
printf("\n\tYou will next enter the Campbell Scientific, Inc. TELCOM\n");
printf("program to edit the datalogger communications file.  Make sure\n");
printf("the proper datalogger is chosen, the data file format is comma\n");
printf("delineated ASCII, the proper communications modem is chosen, and\n");
printf("the 'time for next call' is set to a time or date later than the\n");
printf("present (or you will waste time calling the station after editing");
printf("\nthe communications file).\n");
```

```
   printf("\tMove through the options by pressing 'enter' and move \n");
   printf("through the choices in an option by pressing the space bar.\n");
   printf("\tWhen you are finished editing, press ctrl-P to save the info\n");
   printf("and exit.");
   printf("\n\n\t\t(press any key to continue)");
   getch();
   printf("\n\n");
   spawnl(P_WAIT, "telcom", "telcom", info.sta_name, "/e", NULL);

   return;
}  /* end EditProgramInfo */


void InitializeProgramInfo(void)
/*-------------------------------------------------------------------------*/
/* Initializes program information.                                        */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   strcpy(info.sta_name, "sta");  /* Weather station name */
   strcpy(info.raw_file, "sta.dat");  /* Data retrieved from station */
   strcpy(info.prev_file, "sta90.pr");  /* Previous data for qc */
   strcpy(info.backup_file, "sta90.bu");  /* Backup hour,day,prec data */
   strcpy(info.current_file, "sta90.cu");  /* Current data, IBSNAT format */
   strcpy(info.curgen_file, "osst0912.w90");  /* Current/gen'd IBSNAT data */
   strcpy(info.qc_file, "sta90.qc");  /* File for QC values */
   strcpy(info.location_id, "os");  /* Location id for IBSNAT file */
   strcpy(info.station_id, "st");   /* Weather station id */
   info.lat = 44.35;  /* Weather station latitude */
   info.lon = 123.10;  /* Weather station longitude */
   info.parfac = 12.07;  /* Conversion factor to change MJ/m2 to umoles/m2 */
   info.pardat = 0.0;  /* Indicates if PAR data is included in weather file */
                       /*    Set to zero if none, set to one if included */
   info.year = 90;  /* Year weather data collection starts */
}  /* end InitializeProgramInfo */


void MonitorStation(void)
/*-------------------------------------------------------------------------*/
/* Allows user to monitor weather station in operation, and to set the     */
/*  datalogger clock.                                                      */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */

   /* See if want to monitor weather station */
   while(TRUE){
       _clearscreen(_GCLEARSCREEN);
      printf("\nYou will be entering the Campbell Scientific, Inc. TERM");
      printf("\n   program.");
      printf("\nTo monitor the station, choose the 'monitor' option.");
      printf("\nTo set the time and date in the station's datalogger, choose");
      printf("\n   the 'set datalogger clock' option.  Before choosing this");
      printf("\n   option, make sure the time and date are properly set on");
      printf("\n   this computer.");
      printf("\n\nAre you sure you want to continue? ");
         choice = getche();
         if((choice == 'y') || (choice == 'Y'))
            break;
         if((choice == 'n') || (choice == 'N'))
            return;}

   spawnl(P_WAIT, "term", "term", info.sta_name, NULL);

   return;
}  /* end MonitorStation */
```

```
void PrintProgramInfo(void)
/*-------------------------------------------------------------------------*/
/* Displays program information on screen.                                 */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: none.                                                            */
/*-------------------------------------------------------------------------*/
{
   _clearscreen(_GCLEARSCREEN);
   printf("\n\tWeather station name:  %s", info.sta_name);
   printf("\n\t\tPrevious data file name:  %s", info.prev_file);
   printf("\n\t\tBackup data file name:  %s", info.backup_file);
   printf("\n\t\tCurrent IBSNAT data file name:  %s", info.current_file);
   printf("\n\t\tQC data file name:  %s", info.qc_file);
   printf("\n\t\tRaw data file name:  %s", info.raw_file);
   printf("\n\t\tDatalogger communications file name:  %s.stn", info.sta_name);
   printf("\n\tCurrent & generated IBSNAT data file name:  %s",
         info.curgen_file);
   printf("\n\tLocation ID:  %s", info.location_id);
   printf("\n\tWeather station ID:  %s", info.station_id);
   printf("\n\tLatitude: %6.2f", info.lat);
   printf("\n\tLongitude: %6.2f", info.lon);
   printf("\n\tCurrent year: %d", info.year);
   printf("\n\n");
   return;
}  /* end PrintProgramInfo */


void StartNewDataYear(void)
/*-------------------------------------------------------------------------*/
/* Allows user to begin a new year of data collection.                     */
/* Receives: none.                                                         */
/* Globals:  info                                                          */
/* Returns: none.                                                          */
/* Calls: EditProgramInfo                                                  */
/*-------------------------------------------------------------------------*/
{
int choice; /* user's choice */

   /* Ask if want to proceed */
   while(TRUE){
      _clearscreen(_GCLEARSCREEN);
      printf("\n\nThis option will stop the placement of weather data into");
      printf(" the\n   present files and will create new files for all future");
      printf(" data.");
      printf("\n\n\tAre you sure you want to do this? ");
      choice = getche();
      if((choice == 'y') || (choice == 'Y')){
         printf("\n\n");
         break;}
      if((choice == 'n') || (choice == 'N'))
         return;}
   remove(info.prev_file);
   remove(info.curgen_file);
   rename(info.current_file, info.curgen_file);
   EditProgramInfo(NEW_DATA_YEAR);
   return;
}  /* end StartNewDataYear */
```